Tomi Turunen

Tehonvalvontasovelluksen kehitys

Metropolia Ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka
Insinöörityö
15.11.2011

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Tomi Turunen Tehonvalvontasovelluksen kehitys 43 sivua + 1 liite 15.11.2011
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Simo Silander Lead Engineer Juha Mäntysaari

Insinöörityössä käsitellään tehonvalvontasovellusta ja yritysten tehonvalvontaa. Tavoitteena työssä oli ABB Oy prosessiteollisuuden energiahallintajärjestelmän (cpmPlus Energy Manager) tehonvalvontasovelluksen kääntäminen C++-ohjelmointikieleltä C#-ohjelmointikielelle sekä uusien ominaisuuksien kehittäminen sovellukseen.

Tehonvalvontasovellus on osa ABB Oy:n energianhallintaohjelmistoa (cpmPlus Energy Management), jota käytetään lähinnä suurissa prosessiteollisuuden tehtaissa esimerkiksi metalli-, paperi- ja sähköteollisuudessa. Käyttämällä tätä ohjelmistoa yritys voi säästää sähkönhankintakuluissa. Tehonvalvontasovelluksella valvotaan yrityksen osaston loistehon, pätötehon tai kaasun käyttöä energiayhtiöiden asettamia rajoja tai itseasetettuja ennusteita vasten. Tehonvalvontasovelluksessa käytetään apuna ABB Oy:n kehittämää RTDB-tietokantaa ja Vtrin-käyttöliittymää.

Sovellus saatiin käännettyä C#-ohjelmointikielelle ja lisättiin uusia ominaisuuksia kuten ennusteen laskeminen pienimmän neliösumman menetelmällä ja hälytysrajojen asettaminen jakson keston tai ylityksen suuruuden mukaan. Ohjelmaa kehittäessä tuli uusia ideoita ominaisuuksista, kuten uusien tietokantataulujen lisäys. Nämä uudet ideat tullaan toteuttamaan ja ne liitetään sovellukseen ennen kuin tuote otetaan käyttöön asiakasjärjestelmissä. Sovellus julkaistaan, kun ominaisuudet on lisätty ja sitä on testattu tarpeeksi kattavasti.

Avainsanat	C#, tehonvalvonta, tietokannat

Metropolia

Abstract

Author(s) Title Number of Pages Date	Tomi Turunen Tie-Line Monitoring 43 pages + 1 appendix 15 November 2011
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer Juha Mäntysaari, Lead Engineer

This Bachelor's thesis deals with Power Management Application and business performance monitoring. The aim was to work with an ABB industrial energy management system (cpmPlus Energy Manager) Power Control Application by compiling it from C++-programming language to C# and to develop the application further by adding new features to it.

Power Management Application is part of ABB's Energy Management Software (cpmPlus Energy Management), which is used mainly in large industrial process plants such as metal, paper and electrical industries. Using this software, the company can save in energy purchase costs. The Power Management Applications are used to monitor the use of reactive power, active power or gas energy against the limits set by energy companies, or against self-set predictions. The Power Management Application is used to help ABB develop a database RTDB's and the user interface Vtrin.

The application was translated into C# with new features added, such as the forecast calculation with the least squares method and setting the alarm levels according to the duration of the period or the magnitude of the overshoot. During the development of the program new ideas for features, such as the addition of new database tables, were obtained. These new ideas will be implemented and assigned to the application prior to shipping the product to the customers. The application will be published once the features have been added and it has been tested extensively enough.

Keywords	C #, Tie-Line Monitoring, databases

Sisällys

KÄSITTEET JA LYHENTEET

1 Johdanto 1

2 Työn esittely 2

2.1 Energianhallintajärjestelmän ja tietokannan esittely 2
2.1.1 Energianhallinta 2
2.1.2 Järjestelmä 3
2.1.3 Tietokanta 4
2.1.4 Käyttöliittymä 6
2.2 Käännöstyön esittely 7
2.2.1 Tehonvalvontasovellus 7
2.2.2 Tehonvalvonnan käyttötarkoitukset 8
2.2.3 Vanha sovellus 10
2.2.4 Kääntämisen etuja 11

3 Käännösprosessin toteutus 13

3.1 Luokkien toteutukset 13
3.2 Tietokantaoperaatiot 15
3.2.1 Kannasta lukeminen 15
3.2.2 Kantaan kirjoittaminen 18
3.2.3 Aikojen käsittely 20
3.3 Historia 21
3.4 Ennustaminen 22
3.5 Raja-arvojen ja jakson laskutoimitukset 23
3.6 Lokitulostus 25

4 Uudet ominaisuudet 26

4.1 Pienimmän neliösumman menetelmä 26
4.2 Hälytysrajat 29
4.2.1 Hälytysrajat kuluneen ajan mukaan 29
4.2.2 Hälytysrajat ylityksen mukaan 30
4.3 Valvontarajojen otto historiataulusta 30

5 Testiajot 31

5.1 Testiajo 1: Kumulatiivinen laskutapa 31
5.2 Testiajo 2: Alitus 34
5.3 Testiajo 3: Pienimmän neliösumman menetelmä 35
5.4 Testiajo 4: Tasaisten hetkellisarvojen tapaus 37
5.5 Testiajo 5: Kuukausikohtainen ennuste 38

6 Jatkokehitysideoita 40

7 Yhteenveto 40

Lähteet 42

Liitteet

Liite 1. Esimerkkikoodi 2: Historiatieto- ja ennustetaulujen kirjoitus tietokantaan kokonaan

KÄSITTEET JA LYHENTEET

.NET	Microsoftin kehittämä ohjelmistokomponenttikirjasto, jota käyttävät Microsoftin VisualStudio.NET-ympäristössä kehitetyt ohjelmistot.
C++	1980-luvulla C-kielestä kehitetty ohjelmointikieli, joka on yksi tämän hetken tärkeimmistä kaupallisessa ohjelmistokehityksessä käytettävistä ohjelmointikielistä.
C#	Microsoftin kehittämä ohjelmointikieli, joka kehitettiin yhdistämään C++:n tehokkuus ja Java-kielen helppokäyttöisyys.
cpmPlus Energy Manager	Teollinen tietojärjestelmä, joka on suunniteltu silmällä pitäen tehokasta energian käytön hallintaa ja optimointia.
Delphi	Borlandin tekemä Object Pascal -kieleen perustuva ohjelmointikieli ja graafinen ohjelmankehitysympäristö.
dotGNU	GNU Projectin vastine .NET-kehykselle.
EM	Energy Management. ABB Oy:n luoma energianhallintajärjestelmä.
ISO	International Organization for Standardization, kansainvälinen standardisoimisjärjestö.
Java	Laitteistoriippumaton ja helppokäyttöinen ohjelmointikieli.
Modula 2	1970-luvulla kehitetty nykyään enää harvoin käytetty ohjelmointikieli.
Mono	Ohjelmistokehitysympäristö.
RTDB	ABB Oy:n kehittämä relaatiotietokanta.
Smalltalk	Ohjelmointikieli ja ohjelmointiympäristö, jota pidetään Javan edeltäjänä.
Tehonvalvontasovellus	Yksi osa ABB Oy:n cpmPlus Energy Manager -ohjelmistoa.
Turbo Pascal	Pascal-ohjelmoitikieltä varten kehitetty ohjelmankehitysympäristö 1980-luvulta.
Vtrin	IT-teollisuuden käytössä oleva käyttöliittymä, jolla voi muun muassa seurata arvoja ja käyriä.

XML	*eXtensible Markup Language*. Standardi, jolla tiedon merkitys on kuvattavissa tiedon sekaan.
XSD	*XML Schema Definition*. XML Scheman käyttösovellus.

1 Johdanto

Työn tavoitteena on ABB Oy prosessiteollisuuden energiahallintajärjestelmän (cpmPlus Energy Manager) tehonvalvontasovelluksen kääntäminen C++-ohjelmointikieleltä C#-ohjelmointikielelle sekä uusien ominaisuuksien kehittäminen sovellukseen. Uusia ominaisuuksia sovellutuksessa ovat muun muassa ennustaminen pienemmän neliösumman sovitteen avulla, jonka tarkoituksena on pyrkiä mahdollisimman tarkkaan ennustukseen, ja hälytysrajat. Hälytysrajat ovat rajoja, joiden sisällä ei anneta hälytystä rajojen ylittämisestä. Jos hälytys on pieni tai tapahtuu aivan jakson alussa, niin hälytystä ei ole välttämättä tarpeen tehdä.

Sovellus tallentaa tulokset RTDB (real time database, cpmPlus History) -tietokantaan ja sitä käytetään tietokannan Vtrin-käyttöliittymän avulla. Käyttöliittymällä tehdään sovelluksen määrittelytyöt ja tulosten seuranta. ABB Oy:llä on ollut jo vuosia asiakaskäytössä C++-ohjelmointikielellä toimiva ohjelma, mutta sen toiminta halutaan nykyaikaistaa, ylläpitoa helpottaa ja kehittää uusia ominaisuuksia.

Työssä tutustutaan myös Energy Manager- järjestelmään, sekä siihen kuuluvaan tehonvalvontasovellukseen ja sen toimintaan, käyttötarkoituksiin, sen tuomiin hyötyihin sekä käyttöönottoon yrityksissä. Lisäksi tutustutetaan lukija myös tehonvalvontasovelluksen käyttämään tietokantaan ja käyttöliittymään. Myös uuden tehonvalvontasovelluksen koodin rakennetta ja toiminnallisuutta käydään läpi. Tämän ohella käydään läpi käyttöliittymäkuvia, joista käy ilmi sovelluksen käyttäjälle näkyvä toiminta eri tilanteissa. Lopuksi esitetään tapoja, miten jatkossa voidaan kehittää sovellusta palvelemaan entistä paremmin asiakkaiden käyttötarpeita.

2 Työn esittely

2.1 Energianhallintajärjestelmän ja tietokannan esittely

2.1.1 Energianhallinta

Tehonvalvontasovellus on osa ABB Oy:n cpmPlus Energy Manager -ohjelmistoa. cpmPlus Energy Manager (EM) -järjestelmä on teollisuuden käyttämä tietojärjestelmä, joka on suunniteltu silmällä pitäen teollisuusyritysten tehokasta energian käytön hallintaa ja optimointia. Järjestelmää käytetään lähinnä suurissa prosessiteollisuuden tehtaissa, esimerkiksi metalli-, paperi- ja sähköteollisuudessa. Järjestelmää käyttävät esimerkiksi Pohjolan Voima ja UPM-Kymmene. Teollisuusyritykset saavat aikaan merkittäviä säästöjä energiakustannuksissa käyttämällä järjestelmän valvonta- ja ennustetyökaluja sekä energian käytön ja hankinnan optimointia. [1, s. 9.]

EM-järjestelmän tärkeimmät osa-alueet ovat

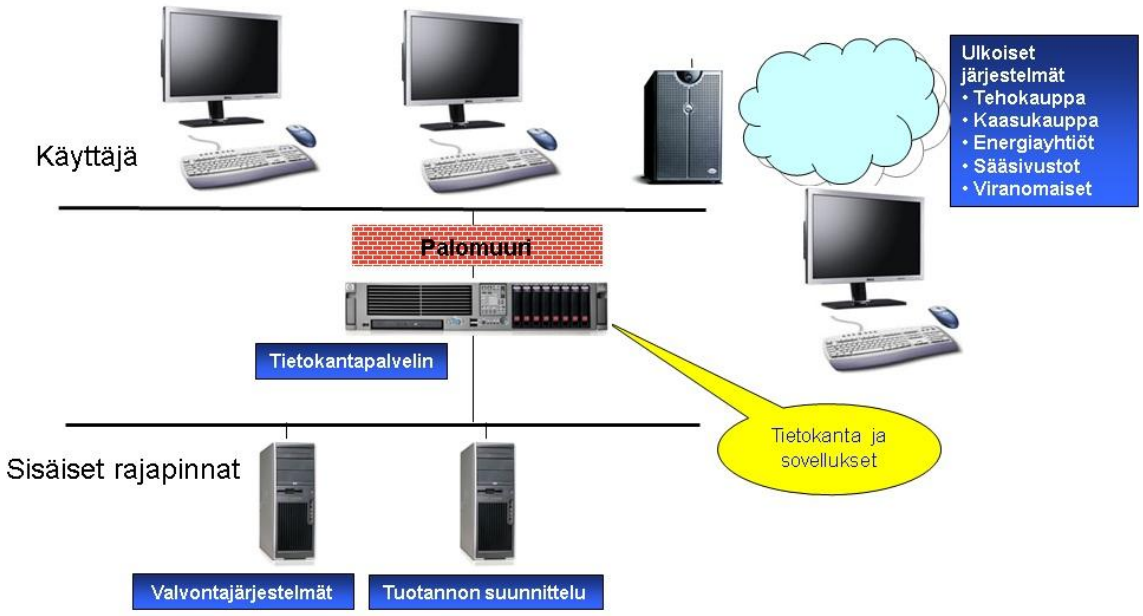
- energiankulutuksen ennustaminen
- tuotannonsuunnittelu tai resurssien optimointi, jonka avulla tehdään kustannustehokas energiaresurssien käyttösuunnitelma ennustetun kuorman mukaan
- tehonvalvonta, jossa energian käyttöä verrataan sopimuksellisiin rajoihin kuluvalla sähkön laskutusperiodilla
- energiaresurssien käytön selvitysprosessi perustuen todellisiin mittauksiin
- visualisointi, laskenta ja raportointi.

Näistä monista osa-alueista EM-järjestelmään voi kuulua yksi, useita tai kaikki edellä mainitut. [1, s. 9.]

Kulutuskohdekohtainen käytön ennustaminen voi koskea sähkötehoa, höyryvirtaa, kaasuvirtaa sekä valinnaisesti myös muita kokonaisuuksia yhdellä aikatasolla. Tyypillinen käyttökohde on tehdas, jolla on useita erilaisia tuotanto-osastoja. Tehdas on jaettu suurimpiin kohteisiin, joiden kulutus on mahdollista ennustaa erilaisilla menetelmillä. [1, s. 10.]

2.1.2 Järjestelmä

Tehonvalvontasovellus pyörii Windows-sovelluksena tietokantapalveluna. Asiakkaalle toimitetaan tietokantapalvelin, johon on tietokanta ja tilatut EM-järjestelmän osat asennettu. Järjestelmää käytetään client PC:iltä, joille Windows 7 on suositeltu käyttöjärjestelmä, mutta järjestelmä saattaa toimia vanhemmillakin Windows-versioilla. Linuxilla tai Macintoshilla järjestelmä ei toimi.



Kuva 1: Järjestelmäarkkitehtuuri [2, s. 6.]

Kuvasta 1 näemme, miten EM-järjestelmän arkkitehtuuri muodostuu. EM-järjestelmä, johon tehonvalvontasovelluskin kuuluu, pyörii Windows-palveluna tietokantapalvelimella yhdessä tietokannan kanssa. Asiakkailla on omia automaattisia valvontajärjestelmiä, jotka muun muassa mittaavat tehonkulutusta tehtaassa. Tietokantapalvelin hakee tiedot valvontajärjestelmistä ja tallentaa ne historiatauluihin automaattisesti. Järjestelmää käytetään erillisellä tietokoneella Vtrin-käyttöliittymästä käsin.

2.1.3 Tietokanta

Tietokantana sovelluksessa käytetään ABB Oy:n kehittämää RTDB (Real Time Database) -tietokantaa. RTDB on relaatiotietokanta ja siinä on vakiona käyttöliittymäkirjasto, jonka avulla sovellukset voivat kirjoittaa tai lukea tietoja tietokannasta. Lisäksi RTDB-tietokanta toimii yhdyskäytävänä muiden tietokantojen kuten Oraclen tietoihin. RTDB on suunniteltu korkean suorituskyvyn omaaviin tietojärjestelmiin ja sen joustavuus, yleinen toimivuus ja modulaarinen rakenne tekevät siitä optimaalisen tietokannan lukuisiin eri sovelluksiin teollisuuskäytössä. RTDB tarjoaa relaatiotietokannan, joka on optimoitu ja suunniteltu teollisuusprosessien tietohallintoon ja laajaan historian tallennukseen. [7, s. 18.]

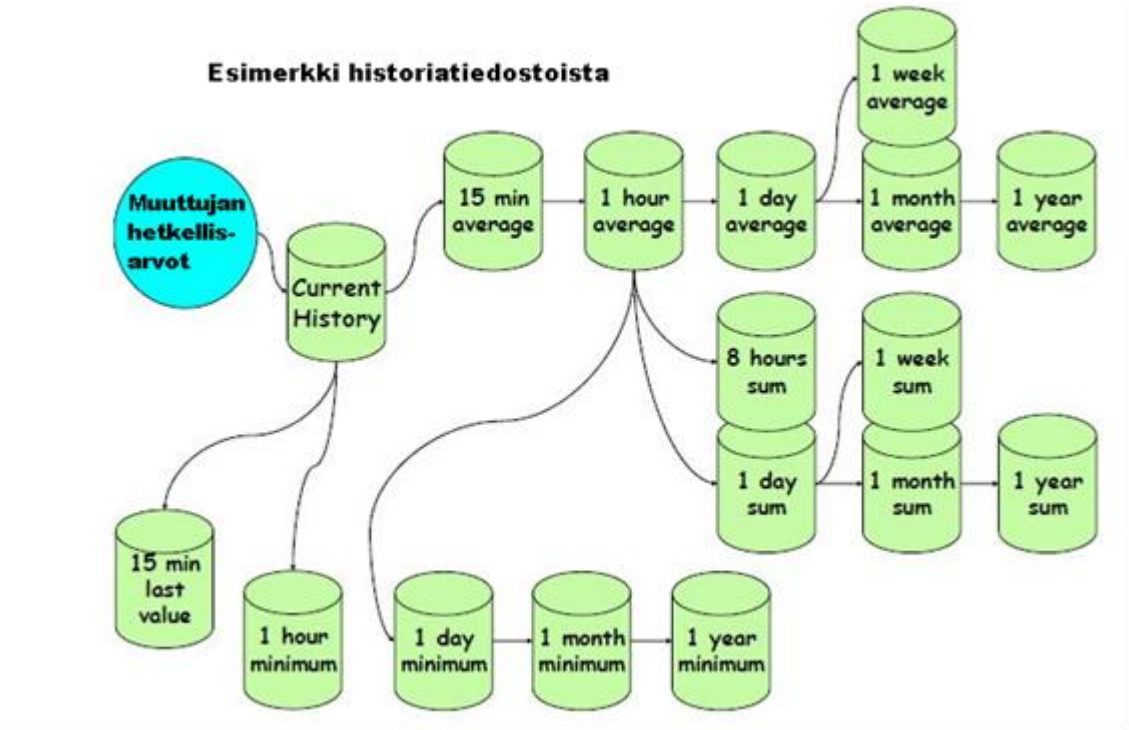
Tehokkuus ja luotettavuus yhdessä huoltovapaan toiminnan kanssa takaavat vankan pohjan tärkeille järjestelmille. RTDB tarjoaa myös työkaluja erilaisten ratkaisujen rakentamiseen, jossa kaikki eri järjestelmien toiminnot on toteutettu RTDB-tuoteperheeseen. Avoimuus ja standardiliitännät tekevät siitä myös helposti integroitavan sekä kolmannen osapuolen tuotteisiin että muihin RTDB-pohjaisiin ratkaisuihin. [7, s. 18.]

Hyvä esimerkki RTDB:n sovellusalueesta on teollisuuden energianhallintajärjestelmien suunnittelu, seuranta, valvonta ja raportointi. Myös muilla aloilla kuten esimerkiksi metalli-, paperi-, öljy-, kaasu-, kemikaali- ja voimalaitoksissa RTDB:tä on käytetty muun muassa seurantaan, valvontaohjaukseen, optimointiin ja prosessien analysointiin. Näiden lisäksi RTDB:tä on käytetty myös prosessiteollisuuden tuotannon suunnitteluun ja johtamisjärjestelmiin. [1, s. 10; 7, s. 18.]

RTDB pyrkii olemaan yhtä joustava kuin yleiskäyttöiset tietokannat ja samalla yhtä suorituskykyinen kuin patentoidut perinteiset yksilöllisemmät tietokannat. RTDB sisältää sovelluskohtaiset taulut ja relaatiot. Tämän lisäksi tietokannassa voidaan luoda myös ennalta määriteltyjä tietokantatauluja. Muuttujia voi myös tunnistaa esimerkiksi tagien, arvojen tai tallennusajankohdan mukaan tietokannasta. RTDB sisältää valmiit toiminnot muun muassa mittauksen esikäsittelyyn sekä hälytyksien ja tapahtuminen käynnistämiseen. Mittauksen esikäsittelyssä voidaan skaalata mittaustuloksia lausekkeella tai vakiolla. RTDB voi myös käsitellä laajasti muuttujahistorioita sisäänrakennetun tilaa säästävän historian pakkauksen ansiosta. Lisäksi RTDB laskee

tilastollisia suureita kuten esimerkiksi ajan keskiarvoja ja summia sekä minimi- ja maksimiarvoja. [7, s. 19.]

Tehonvalvontasovellus käyttää tietokannan historiatauluja hakeakseen sieltä tarvitsemansa historiatiedot. Järjestelmä kerää hetkellisarvot ja laskee niistä keskiarvot, summat ja minimit eri aikaväleille. [3, s. 27.]



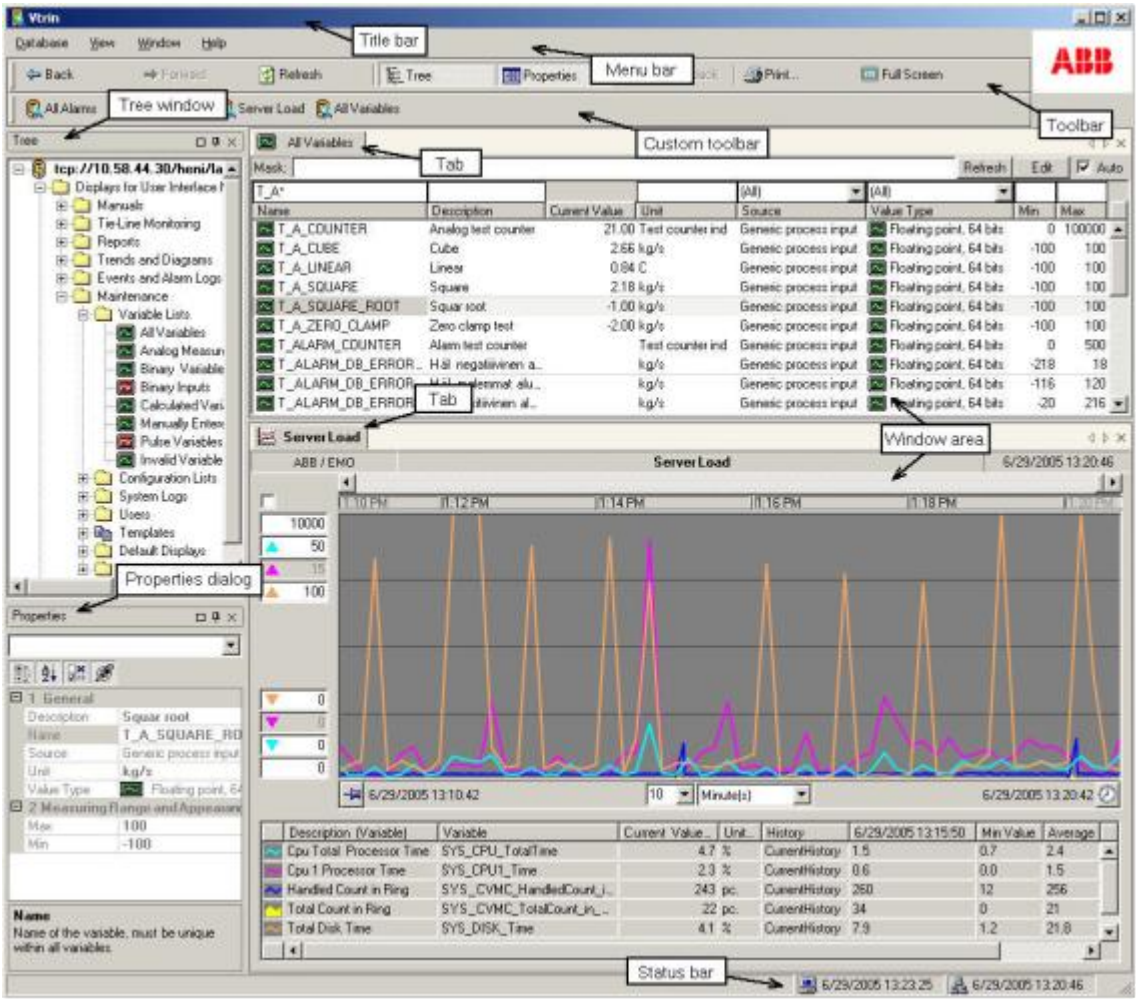
Kuva 2: Historiataulut [3, s. 28.]

Kuvasta kaksi käy ilmi, että tietokannasta löytyy keskiarvoja (avarage), summaa (sum) ja minimiarvoja (minimum) kuvaavia historiatauluja. Kuvasta ilmenee myös miten historiataulut muodostetaan. Esimerkiksi keskiarvo lasketaan 15 minuutille, tunnille, päivälle, viikolle, kuukaudelle ja vuodelle. Tarvittaessa historiatauluja voi myös tehdä lisää. Tehonvalvontasovelluksessa käytetään yleisimmin Current Historya, kun valvotaan esimerkiksi tehon kulutusta yhden tunnin ajalta. Tällöin haetaan siis raaka-arvot, ei keskiarvoja. Kun taas mitataan pidempää aikaa, kuten kuukautta tai vuotta, haetaan arvo tunnin tai päivän keskiarvoista. Muuttujan hetkellisarvot saadaan tehtaan automaattisista valvontajärjestelmistä, jotka mittaavat niitä. [3 , s. 27-28.]

2.1.4 Käyttöliittymä

Tehonvalvontasovelluksen käyttöliittymä on Vtrin. Vtrin on ABB Oy:n kehittämä IT-teollisuuden käytössä oleva käyttöliittymä, joka voidaan yhdistää RTDB-tietokantoihin, ja IT-teollisuuden järjestelmistä tuotannon suunnitteluun, tuotantoon, laadunhallintaan sekä energian hallinnointiin ja optimointiin. [8.]

Tehonvalvontasovellukselle syötetään Vtrinin konfigurointiominaisuuden avulla parametrit, jotka siis käyttöliittymä syöttää tietokantaan. Vtrin tarjoaa joukon vakionäyttöjä datalle ja työkalut sovelluskohtaisten näyttöjen luomiseen. Näytön asetukset voidaan konfiguroida verkon yli käyttäjäystävällisten drag & drop -valikkojen avulla. Standardinäytöt voivat sisältää muun muassa reaaliaikaisia ja historiallisia kuvaajia. Kuvaajia voi piirtää eri muodoissa kuten trendikäyrinä, sarakkeina tai Gantt-kaavioina. Käyttäjä voi määritellä kuvaajissa esitetyt tiedot kuten esitettävä datan, esitettävän aikavälin ja kaavion tyypin. Lisäksi hetkellisarvoja voi tarkkailla alaruudussa olevasta hetkellisarvosarakkeesta (Current Value). Vtrin ei itse laske mitään, vaan piirtää kuvaajat tietokannasta löytyvien tietojen mukaan. Vtrin vaatii voimassa olevat tunnukset ja sisäänkirjautumisen. [3, s. 30.]



Kuva 3: Vtrinin näyttö [8, s. 9.]

Kuvassa kolme näkyy Vtrinin näyttö eriteltynä alueittain. Kuvan keskellä näkyy kuvaaja ja sen yläpuolelle on listattu muuttujia. Ruudun alaosassa on seurattavat käyrät ja hetkellisarvot (Current Value). Vasemmassa nurkassa olevasta puurakenteesta käyttäjä saa valittua haluamansa kaavion. Uusien kaavioiden luonti onnistuu painamalla oikeaa hiirennäppäintä ja valitsemalla new tree item ja sieltä haluamansa kaaviotyyppi, kuten esimerkiksi trendi (Trend).

2.2 Käännöstyön esittely

2.2.1 Tehonvalvontasovellus

Tehonvalvonta alkoi siitä, kun energiayhtiöt olivat asettaneet yhtiöille maksimitehorajoja, joissa tuli pysyä. Jos raja ylitettiin, jouduttiin maksamaan

enemmän. Myös silloin jos ostettiin liian suuri tehoraja, jouduttiin maksamaan ylihintaa, koska olisi pärjätty pienemmälläkin tehomäärällä. Tehonvalvonnan historia alkoi pätötehon valvonnasta. Nykyjään yleisempi käyttökohde on valvoa kuluvan periodin tehonkulutusta vasten kyseiselle periodille annetun sähkönkäytön ennustetta.

ABB Oy:llä on ollut jo useita vuosia asiakaskäytössä C++-ohjelmointikielellä tehty tehonvalvontasovellus. Tehonvalvontasovellus on yksi osa ABB:n cpmPlus Energy Manager -ohjelmistoa. Tehonvalvontatoimintoa voi käyttää minkä hyvänsä jatkuvan muuttujan, kuten esimerkiksi ostetun sähkön tai kaasun määrän valvontaan vasten annettuja rajoja. Pylväskaavionäytöt helpottavat järjestelmän valvontaa. Pylväskaavioina näkyvät ostetun sähkön tai kaasun määrät. Niiden ekstrapolointirivi ilmoittaa arvioidun kokonaismäärän valvontajakson lopussa. [1, s. 39.]

Tämän lisäksi kaaviosta tulee ilmi myös paljon muita hyödyllisiä tietoja, kuten esimerkiksi valvontarajojen ylitys tai alitus jakson lopussa. Kun sovellus huomaa, että ylitys tai alitus ylittää hälytysrajan, voidaan tehdä ilmoitus tietokannan sovellukselle, joka tekee hälytyksen. Kun hälytys on annettu, tehonvalvontasovellusta käyttävät valvojat voivat ryhtyä parhaaksi katsomiinsa toimenpiteisiin, mikäli katsovat tämän tarpeelliseksi. Näin yritys välttyy ylimääräisiltä maksuilta, jota seuraisi rajan liian suurista ylityksistä tai alituksista. Yleinen hälytystä seuraava toimenpide on kulutuksen rajoittaminen ja toinen usein käytetty mahdollisuus on vaihtoehtoisten energiavaraintojen käyttöönotto, joihin päädytään yleensä silloin, kun resurssit eivät riitä tai ennakoidun marginaalin kustannukset tulevat liian kalliiksi.[1, s. 39; 3, s. 11.]

2.2.2 Tehonvalvonnan käyttötarkoitukset

Kuluttajien sähköverkosta ottama sähköteho voidaan jakaa kahteen eri osaan. Esimerkiksi nyt jo EU-alueella käytöstä poistettujen hehkulamppujen toiminta perustuu siihen, että sähkövirta saa vastuslangan lämpenemään ja hehkumaan. Hehkulamppu siis ottaa verkosta energiaa sähköisessä muodossa ja muuttaa sen muiksi energiamuodoiksi, eli lämmöksi ja valoksi. Sähköenergiaa käytetään usein tekemään myös mekaanista työtä, kuten esimerkiksi liikuttamaan sähkömoottorin avulla hissiä. Tällaisen energiankäytön tehoa kutsutaan pätötehoksi. Pätötehoa voidaan pitää myös

todellisuudessa kulutettuna tehona, koska siinä energiaa siirtyy pois sähköverkosta sen ympäristöön. [5; 6.]

Useiden sähkölaitteiden toiminta edellyttää myös energian väliaikaista varastoimista esimerkiksi laitteiden sisältämiin keloihin. Jo mainittujen sähkömoottorien lisäksi tällaisia laitteita ovat muun muassa vanhat loistevalaisimet sekä muuntajat. Tämä varastoitunut energia on otettava sähköverkosta, ja palautuu sinne takaisin, koska sitä ei suoranaisesti kulutetakaan mihinkään, kuten muuteta lämmöksi tai käytetä työn tekemiseen. Tällaisen energian siirron yhteydessä puhutaan loistehosta. Sähköyhtiöiden on varauduttava myös loistehon tuottamiseen ja toimittamiseen. Käytännössä loistehokin edustaa sähköntuottajalle aina energiamenekkiä sähköverkon siirtolinjojen lämpöhäviöiden vuoksi. Loistehon siirtotarvetta voidaan kuitenkin vähentää paikallisten, kuluttajaa lähellä olevien kompensointilaitteiden kuten kondensaattoriparistoiden avulla. Kondensaattoriparistot asennetaan sähkökeskuksiin, mistä käsin ne kytkevät verkkoon kondensaattoreita aina tarvittaessa. [4.]

Tällä hetkellä loistehosta ei vielä pyydetä yksityisiltä talouksilta lisähintaa Suomessa, vaan lasku tulee pätötehon eli todellisuudessa käytetyn tehon mukaan. Kuitenkin jo monet nykyiset sähkömittarit mittaavat myös loistehon käyttöä, joten on todennäköistä, että myös kotitalouksia aletaan laskuttaa loistehosta tulevaisuudessa, koska se on sähköyhtiöille lisäkustannuksia tuova palvelu.

Tehonvalvonta lähti liikkeelle pätötehon valvonnasta. Tämä olikin ennen tehonvalvontasovelluksen tärkein käyttötarkoitus. Kuitenkin pätötehon merkitys teollisuuden yrityksissä on vähentynyt johtuen muuttuneista energiasopimuksista, joissa laskutetaan erityisesti loistehon perusteella. Nykyään onkin siirrytty enenevissä määrin loistehon valvontaan. Yleisimmin pätötehon ja loistehon kulutusta seurataan tunnin mittaiselta jaksolta, mutta myös vuorokauden mittainen mittausjakso on tavanomainen.

Kolmas merkittävä tehonseurannan kohde on kaasun kulutus, jota seurataan välillisesti. Tehtaat ostavat kaasua säiliöissä tai sitä syntyy tehtaan jonkin toiminnan sivutuotteena. Kaasua poltetaan ja näin siitä saadaan lämpöenergiaa. Tätä lämpöenergiaa käytetään sitten tehtaassa ja tämän energian kulutusta seurataan.

Kaasua voi seurata samaan tapaan käyttöliittymästä kuin tehonkulutustakin. Kaasunkulutuksen seurantaväli on tavallisesti pidempi kuin sähkötehon, yleensä viikosta useisiin viikkoihin.

2.2.3 Vanha sovellus

Vanha tehonvalvontasovellutus on tehty C++-ohjelmointikielellä 2000-luvun alkupuolella. Alkuperäisessä sovelluksessa ajanjakson pituus (period minutes) ilmoitettiin aina minuutteina. Vuonna 2009 sovellukseen tehtiin parannus, jonka jälkeen ajanjakson pystyy ilmoittamaan myös kuukausina. Tämä tapahtuu syöttämällä jakson pituudeksi (period minutes kohta Vtrinissa) kuukausien lukumäärä negatiivisena. Esimerkiksi -12 tarkoittaa siis kahtatoista kuukautta. Tämä ominaisuus lisättiin asiakkaan tarpeesta, kun haluttiin seurata päivä- tai viikkokohtaisia kulutuksia esimerkiksi vuoden ajalta.

Vanha sovellus on jaettu neljään luokkaan: ActicePoweriin, TieLineCalciin, TieLineGetFromBasees ja TieLineMonotoringiin. Näistä TieLineMonitoring on pääluokka. TielineCalc-luokkaa käytetään apuna laskennassa ja TieLineGetFromBase-luokkaan on sijoitettu tietokantaoperaatioita. Laajin luokista on ActicePower, jossa sijaitsevat muun muassa laskutoimitukset ja kirjoitusoperaatiot tietokantaan. Näiden neljän luokan lisäksi sovellus sisältää runsaasti kutsuja metodeihin, jotka sijaitsevat muualla eri projekteissa. Tämän seurauksena vanha koodi oli vaikeaselkoista ja koodista oli vaikea saada selville, mitä missäkin kohtaa tehtiin, koska jouduttiin hakemaan kyseisiä koodeja projektin ulkopuolelta.

Uutta versiota tehtäessä pyrittiin saamaan helppolukuista ja helposti tulkittavaa koodia. Jotta koodi olisi helposti tulkittavaa, kirjoitettiin suurin osa koodista samaan luokkaan. Poikkeuksena oli pienimmän neliösumman menetelmän laskemiseen käytetty luokka APP_LSF. Tietokannan apuna käytössä on myös RTDB_CalcBase-liityntäluokka. Koodista tuli melko pitkä, mutta sen jaksottamiseen käytettiin alueita (region), jolloin koodi säilyi helppolukuisena pituudestaan huolimatta. Lisäksi muutoksena vanhaan käytettiin oliomaista lähestymistapaa. Pääluokka sisältää sisäisen luokan cTieLineMonitor, josta pääluokka luo ilmentymiä ja kutsuu sen metodeita.

Myös tietokantaoperaatiot erosivat huomattavasti uuden ja vanhan version välillä, sillä vanhassa versiossa ne oli tehty varsin monimutkaisesti.

```
#include <RTDB_CVH.h>
#include <RTDB_Time.h>
#include <RTDB_VirtualHistoryTable.h>
#include <CSCommon_StringUtil.h>
#include <CSCommon_StringFormat.h>
#include <CSCommon_Time.h>
#include <EMO_Lib.h>
#include "TieLineData.h"
...

    RTDB::C_LatestValueStatus cls = {0};
  //Producer CR#10-0244
  cls.cls.pr = 51;
      RTDB::C_Time clt;
      clt.M_GetTime();

if ( (*m_pTieLineItems)[NofItem-1].RateToLimitVarIDN > 0 )
 (*m_pTieLineItems)[NofItem-1].RateToLimitDatapoint->
M_ProcessCurrentValue( PowerToLimit, cls, clt );
double FormattedSecondsToLimit = SecondsToMonitLimit / 86400;
if ( (*m_pTieLineItems)[NofItem-1].SecToLimitVarIDN > 0 )
   (*m_pTieLineItems)[NofItem-1].SecToLimitDatapoint-
>M_ProcessCurrentValue( FormattedSecondsToLimit, cls, clt );
```

Koodiesimerkki 1: Vanha koodi

Koodiesimerkissä 1 on esimerkki käännettävästä koodista. Siinä tallennetaan tietokantaan hetkellisarvot. Isolla M-kirjaimella on merkitty ulkopuoliset metodit ProcessCurrentValue ja GetTime. Mistään ei ilmene suoraan, missä kyseiset metodit sijaitsevat, mutta vihjeen siitä antaa sisällyttyjen (include) tiedostojen listaus alussa.

Koodissa on ensin annettu status asettamalla pr-bitti arvoon 51. Aika on saatu GetTime -metodilla. ProcessCurrentValue-metodille annetaan parametreina kirjoitettava arvo, status ja arvon aikaleima ja kyseinen metodi kirjoittaa arvot tietokannan hetkellisarvoon. Tämä tehdään erikseen jokaiselle kirjoitettavalle hetkellisarvolle.

2.2.4 Kääntämisen etuja

C# on Microsoftin .NET-konseptia varten kehittämä ohjelmointikieli. C#-ohjelmointikieli julkaistiin kesäkuussa 2000 ja sen kehitti Anders Hejlsberg, joka tuli Microsoft-yhtiön palvelukseen Borlandilta. Päätavoitteena C#-ohjelmointikielen kehityksessä oli luoda monenlaisiin ympäristöihin soveltuva helppokäyttöinen, oliopohjainen ohjelmointikieli,

jonka kansainvälistäminen olisi myös helppoa. Microsoft sai hankittua C#-ohjelmointikielelle ISO-stardardardin vuonna 2003. Standardin pohjalta on myös tehty ja ollaan tekemässä lukuisia itsenäisiä toteutuksia kielestä, kuten esimerkiksi Mono ja dotGNU. [9; 10; 11; 12.]

C#-ohjelmointikielessä on runsaasti etuja verrattuna perinteiseen C++-ohjelmointikieleen, mistä johtuen tehonvalvontasovelluksen kääntäminen koettiin kannattavaksi. C#-ohjelmointikielessä on automaattinen roskien keruu (Garbage collection), kun C++-ohjelmointikielessä taas ohjelmoija joutuu itse huolehtimaan muistin vapauttamisesta. Yksi C#-ohjelmointikielen hyvä ominaisuus on myös, että siinä voi suoraan osoittaa erillisten binäärimuodossa olevien kirjastojen käytön, toisin kuin C++-ohjelmointikielessä, jossa ne joudutaan ottamaan kääntäessä huomioon (explicit class loading). Yksi näiden kahden ohjelmointikielen välisistä eroista on, että C++-ohjelma voi olla funktionaalinen, kun taas C#-ohjelma perustuu aina olioihin. [13.]

Yksi C#-ohjelmointikielen innovatiivisista toiminnoista on myös komponenttiohjelmoinnin kehittäminen boxing- (muuttaa arvomuuttujan oliomuuttujaksi) ja unboxing- (muuttaa oliomuuttujan arvomuuttujaksi) toiminnoilla. C#-ohjelmointikieli tukee myös XML-tekniikan integrointia ja siinä on myös ohjelmakoodin generointimahdollisuus XSD-skeematiedostosta, joka on XML-pohjainen. Yleisesti ottaen C#-ohjelmointikieli on ohjelmoijalle nopeampaa kirjoittaa kuin C++-ohjelmointikieli, eikä siinä tarvitse käyttää erillisiä header-tiedostoja. C#-ohjelmointikieli auttaa vähentämään ohjelmoinnin virhealttiutta, samalla kun se lisää ohjelmoijan tehokkuutta. [13; 14.]

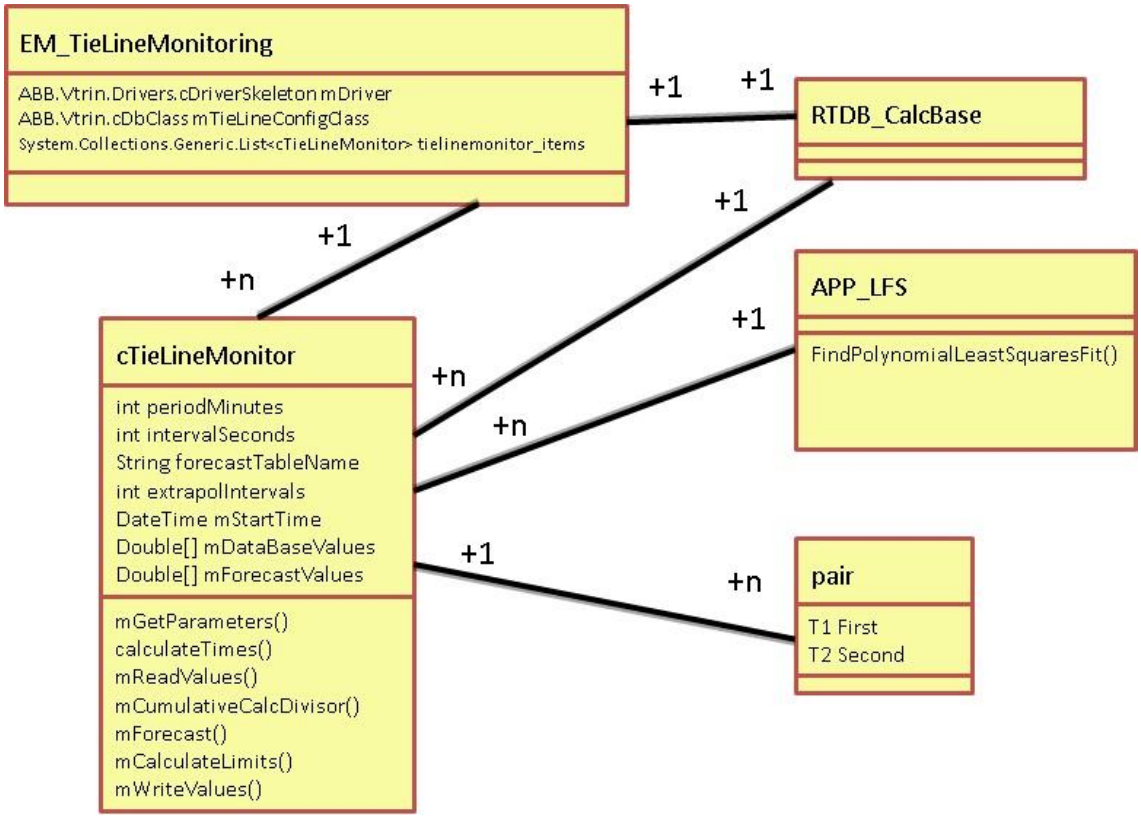
C# on moderni ohjelmointikieli ja se on lainannut jopa niin paljon Javasta, että sitä on kritisoitu Javan kloonikieleksi. Turbo Pascalista ja Borlandin Delphistä aiemmin tunnettu Anders Hejlsberg, joka toimi C#-ohjelmointikielen pääsuunnittelijana kuitenkin kumoaa väitteet. Hejlsberg painottaa, että C#-ohjelmointikielessä on pyritty yhdistämään useita suosittuja ohjelmointikieliä, kuten Modula 2, Java, Smalltalk ja C++ sekä keräämään näistä parhaat puolet yhteen. [15.]

3 Käännösprosessin toteutus

Käännösprosessi C++-ohjelmointikieleltä C#-ohjelmointikielelle toteutettiin käyttäen Microsoft Visual Studio 2008 -ohjelmistoa ja Windows 7 -käyttöjärjestelmää. Visual Studio sopi työhön erityisen hyvin, koska sillä voi kirjoittaa yhtä hyvin kumpaakin ohjelmointikieltä: sekä C++:aa että C#:aa. Sovelluksen testaamiseen käytettiin Vtrin-käyttöliittymää ja Visual Studion Debug-toimintoa.

3.1 Luokkien toteutukset

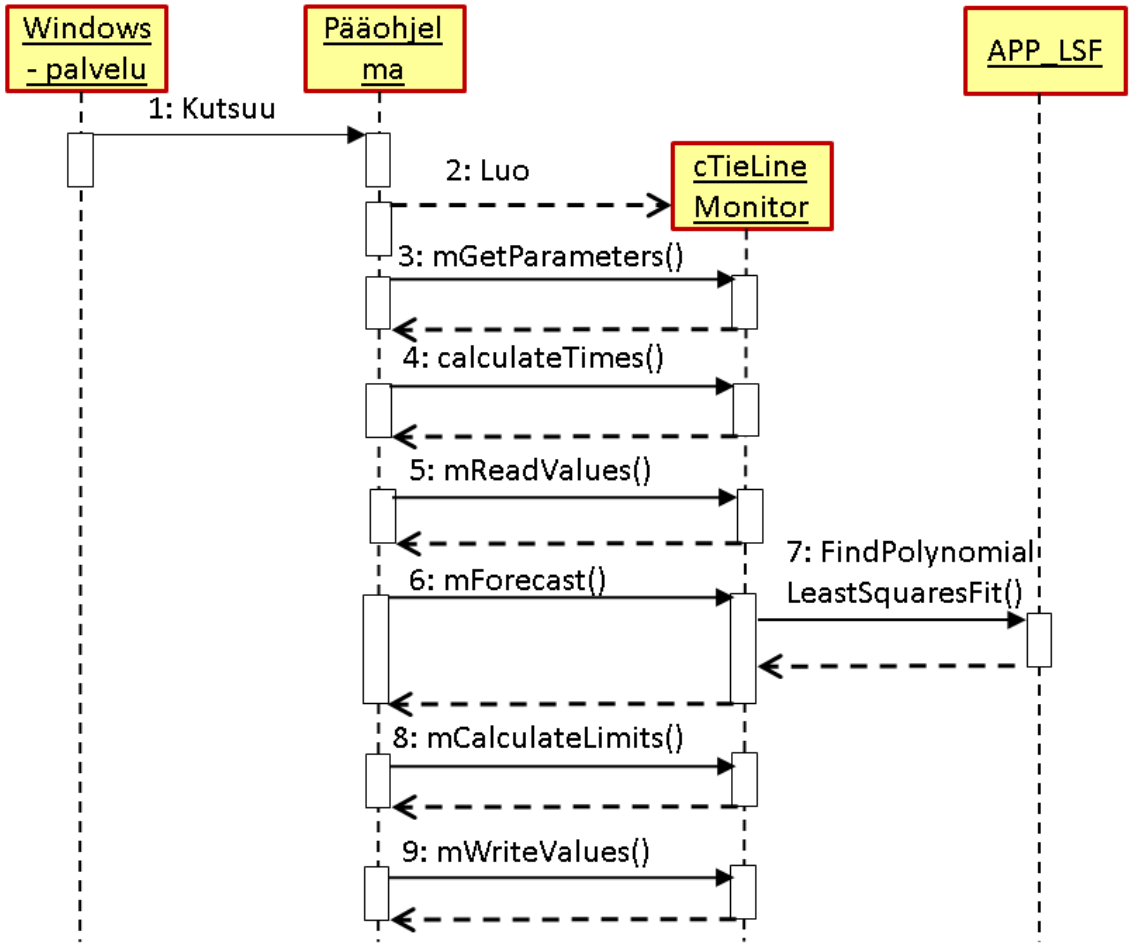
Uusi tehonvalvontasovellus toteutettiin vanhasta versiosta poiketen olioita hyödyntäen. Itse ohjelman suoritus tapahtuu pääluokassa EM_TieLineMonitoring. Itse toiminnallisuus on sisäisessä luokassa cTieLineMonitor. Lisäksi apuna on sisäinen pair-luokka, jonka tarkoitus on korvata C++-ohjelmointikielen pair-muuttujatyyppi.



Kuva 4: Tehonvalvontasovelluksen luokkakaavio

Kuvassa 4 näkyy luokkakaavio tehonvalvontasovelluksesta. EM_TieLineMonitoring luo geneerisen listan (System.Collections.Generic.List), johon lisätään cTieLineMonitor-tyypin olioita, yksi kutakin käyttöliittymän konfigurointiriviä varten. Listaa luodessa

pääluokka merkataan jokaisen luokan cTieLineMonitoring ilmentymän omistajaksi, jotta saadaan pyydettyä myöhemmin pääluokalta tarpeellisia tietoja, kuten ajuri (mDriver) ja luokkainstanssi (mTieLineConfigClass). Kun lista on luotu, suoritetaan jokaiselle listan oliolle yksi kerrallaan operaatiot kutsuen kyseisen olion sisäisiä metodeita.



Kuva 5: Tehonvalvontasovelluksen sekvenssikaavio

Kuvassa 5 on sekvenssikaavio tilanteesta, jossa lasketaan ennuste käyttäen pienimmän neliösumman menetelmää. Pääohjelmasta kutsutaan cTieLineMonitor-olion metodia mGetParameters, joka hakee parametrit tietokannasta. Tietokantaoperaatioita varten tarvitaan myös jakson alkamis- ja päättymisaika sekä nykyhetki. cTieLineMonitor-olion calculateTimes-metodi laskee nämä käyttäen cTieLineMonitor-olion sisäistä mTruncateTime -metodia. Seuraavaksi pääluokka kutsuu metodia mReadValues, joka hakee tietokantatauluista historiatiedot ja kirjoittaa ne muistiin mDataBaseValues-tauluun, joka sijaitsee pääluokassa toistorakenteen sisällä.

Tehojen hetkellisarvoista lasketaan kumulatiivisesti tehon kertyminen jakson alusta nykyhetkeen mCumulativeCalcDivisor-metodilla. Ennuste lasketaan olion mForecast-metodilla antaen sille parametreiksi kumulatiivisesti lasketut tehot, ennustetaulun nimi ja mDataBaseValues-taulu. Lisäksi ennuste kirjoitetaan muistiin väliaikaiseen tauluun (mForecastValues). cTieLineMonitor:in mForecast-metodi kutsuu APP_LFS-luokan metodia, jos ennuste lasketaan pienimmän neliösumman menetelmällä. Kumalatiivisesti laskettaessa APP_LFS-luokkaa ei tavita.

Seuraavaksi suoritetaan cTieLineMonitor-luokan mCalculateLimits- metodin avulla jaksoon ja jaksonrajoihin liittyvät laskutoimituksen mCalculateLimits-metodilla. Sille annetaan parametrina taulut, joissa sijaitsevat hetkellisarvot (mDataBaseValues) ja ennusteet (mForecastValues) . Pair-luokkaa käyttää cTieLine-Olion mCalculateLimits. Lopuksi lasketut tiedot viedään tietokantaan mWriteValues-metodilla. EM_TieLineMonitoring-luokka ja cTieLineMonitor-olio käyttävät molemmat apuna RTDB_CalcBase:a, jota ne tarvitsevat tietokantaoperaatioissa.

3.2 Tietokantaoperaatiot

3.2.1 Kannasta lukeminen

Sovelluksen pitää saada luettua tietokannasta tarvittavat parametrit haluttujen laskuoperaatioiden suorittamista varten. Sovellus tallettaa tietokannasta saamansa parametrit olion sisäisiin muuttujiin, jotta monen eri parametririvin (Vtrin konfigurointirivi) yhtäaikainen laskeminen onnistuu.

Hetkellisarvot tiettyinä ajanhetkinä saadaan annetusta historiataulusta (HistTableName), jonka oletuksena on hetkellisarvo CurrentHistory-taulu. Historiatauluista luku tapahtuu uudessa tehonvalvontasovelluksessa uudella tavalla, josta on olemassa malli ABB Oy:n toisessa energianhallintasovelluksessa. Tätä soveltamalla ja muokkaamalla historiataulun lukeminen saatiin toimimaan tehonvalvontasovelluksessa. Ensimmäisen arvon (first) sijaan tehonvalvontasovelluksessa haetaan historiasta raaka-arvojen keskiarvoa (avgraw).

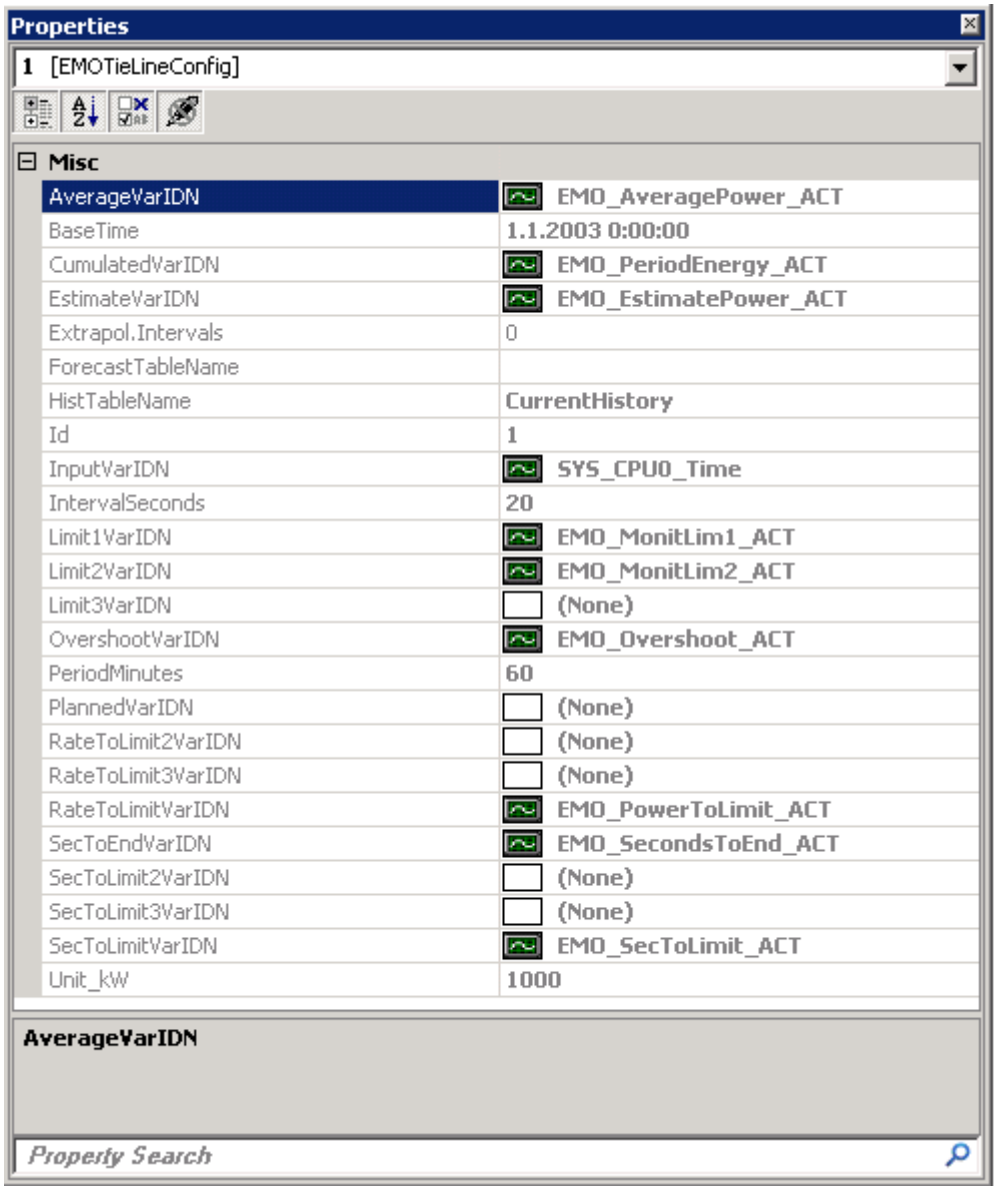
Sovellus tarvitsee myös parametreja. Parametrit syötetään Vtrinin Tieline Configuration -lisänäytöltä. Ensimmäinen syötettävä parametri on PeriodMinutes eli jakson pituus minuutteina. Tämä määrittelee sen, kuinka pitkä on yksi jakso, jonka loppuun lasketaan ennuste. IntervalSeconds, eli väli sekunteina, tarkoittaa, kuinka pitkä on yksi mittausväli. Esimerkiksi jos annetaan arvoksi 60 tarkoittaa se, että mitataan minuutin välein. InputVarIDN kohtaan annetaan muuttuja, mistä hetkellisarvot luetaan. Seuraava kohta PlannedVarIDN kertoo muuttujan, missä sijaitsee jaksokohtainen kulutusennuste. Jos tätä ei ole, voidaan kenttä jättää tyhjäksi.

CumulatedVarIDN-kenttään laitetaan muuttuja, johon lasketaan historialle kumulatiiviset arvot Vtrinin historiapalkkien visualisointia varten. Seuraava kenttä on RateToLimitVarIDN-muuttuja, joka kertoo, kuinka paljon tehoa saa kuluttaa maksimissaan loppujakson ajan, että pysytään vielä ensimmäisen hälytysrajan sisässä. Vastaavasti RateToLimit2VarIDN ja RateToLimit3VarIDN ovat samat toiselle ja kolmannelle hälytysrajalle.

SecToEndVarIDN-kenttään tulee taas muuttuja, johon lasketaan, kuinka monta sekuntia kuluu ennusteen mukaan ensimmäisen rajan ylittämiseen. Vastaavasti SecToLimit2VarIDN ja SecToLimit3VarIDN ilmoittavat toiselle ja kolmannelle rajalle, montako sekuntia ylitykseen ennusteen mukaan on. Jos raja on jo, ylitetty lukee kannassa aikana 00:00:00. SecToEndVarIDN kenttään tulee muuttuja, johon lasketaan, montako sekuntia on periodin loppuun. Kenttään AverageVarIDN tulee muuttuja, johon lasketaan keskiarvot. Kenttään EstimateVarIDN tulee taulu arviolle loppuarvosta ja OvershootIDN-kenttään tulee taulu arvioidulle ylitykselle.

Kenttiin Limit1VarIDN, Limit2VarIDN ja Limit3VarIDN tulevat muuttujat, joista käyvät ilmi asetetut raja-arvot kullakin ajanhetkenä. Unit_kW-kenttään tulee kilowatteina käytetyn yksikön suuruusluokka. Oletuksena on 1000 eli yksi Megawatti. Esimerkiksi vaihtamalla arvoksi ykkösen saa yksiköksi kilowatin. Vastaavasti vaihtamalla arvoksi miljoona tulee yksiköksi Gigawatti. HistTableName-kenttään tulee historiataulun nimi, josta historiatiedot haetaan. BaseTime kohtaan tulee aika, josta jakson ajanlasku aloitetaan. ForecastTableName-kenttään tulee ennustustaulun nimi, jos tällainen on. Tämä on taulu, johon esimerkiksi aikaisempien tietojen perusteella on syötetty arviot tulevista tehonkulutuksista. Tällöin ennustusta ei lasketa, vaan ennustusarvot otetaan

suoraan annetusta taulusta. ExtrapolationIntervals-kohtaan syötetään lukuarvona, kuinka monesta historiatiedosta ennuste lasketaan. Jos parametria ei anneta, säilyy oletusarvona 5.



Kuva 6: Konfigurointi Vtrinin avulla

Kuten näemme kuvasta 6, konfigurointi-ikkunan vasemmassa sarakkeessa on muuttujan nimi ja oikeanpuoleiseen syötetään muuttujan arvo tai muuttuja, mistä arvo löytyy. Tehonvalvontasovellus hakee nämä edellä mainitut arvot tietokannasta.

```
//PeriodMinutes
Object pm = cdbi.GetRawPropertyValue("PeriodMinutes");
if (pm is int)
```

```
{
  PeriodMinutes = System.Convert.ToInt32(pm);
}
```

Koodiesimerkki 2. Parametrin luku tietokannasta

Koodiesimerkistä 2 nähdään, kuinka raaka-arvot haetaan tietokannasta ja tallennetaan olion muuttujiin. Ennen sijoitusta muuttujan tyyppi tarkastetaan virhetilanteiden välttämiseksi.

3.2.2 Kantaan kirjoittaminen

Kantaan kirjoittaminen suoritetaan kirjoittamalla datat tietokantatauluihin ja hetkellisarvoihin RTDB-tietokannassa. Lukuoperaation tavoin myös kantaan kirjoitus uudistui täysin vanhaan versioon verrattuna (vertaa koodiesimerkki 1 ja koodiesimerkki 2).

```
   for (int j = 0; j < data.Length; j++)
   {
   gdi.Write(time, (object)data[j], ABB.Vtrin.cValueStatus.OK |
   ABB.Vtrin.cValueStatus.Representativeness);
   time = time.AddSeconds(IntervalSeconds);
   }
   //time = time.AddSeconds(IntervalSeconds*2);
   for (int j = 0; j < futureData.Length; j++)
   {
    if (j == futureData.Length - 1)
    {
     time = time.AddTicks(-1);
    }
    gdi.Write(time, (object)futureData[j],
    ABB.Vtrin.cValueStatus.OK |
    ABB.Vtrin.cValueStatus.Representativeness);
    time = time.AddSeconds(IntervalSeconds);
   }
  }
 }
 else
 {
  var = null;
  prms = null;
  hist = null;
  gdi.Close();

  return false;
 }
```

Koodiesimerkki 3: Tietokantatauluun kirjoitus

Koodiesimerkissä 3 kirjoitetaan historiatiedot historiatauluun. Tarkastetaan, että tietokantaoperaatio onnistuu varmasti, sillä jos tietokantaoperaatio ei onnistu, ohjelman toiminnan pitää silti jatkua, eikä ohjelma saa kaatua. Tietokantaoperaation epäonnistuessa kirjataan kuitenkin merkintä lokiin. Tietokantaan kirjoitetaan kerralla kaikki parametririvit (Vtrin konfiguraatiorivit). Aloitusaikana on metodille annettu aikaparametri. Sitten annetaan muuttujataulu ja historiataulu tietokantaan kirjoitusta varten ja tarkastetaan, löytyvätkö annettu muuttujataulu ja historiataulu tietokannasta.

Sitten yhdistetään tietokantaan. Jokaisella rivillä on aika, arvo ja status. Aika määräytyy lisäämällä edellisiin aikoihin aina yksi mittausväli. Kirjoitettu data on kumulatiivinen historiatieto ja statukseksi tulee aina ABB.Vtrin.cValueStatus.OK| ABB.Vtrin.cValueStatus.Representativeness. Tämä status kertoo käyttöliittymälle, että arvot ovat kelvollisia. Tiedot kirjoitetaan tietokantaan yksi kerrallaan. Sitten kirjoitetaan vielä futureData-taulusta ennusteet kantaan ja viimeisin arvon ajasta otetaan pois yksi sekunnin kymmenestuhannesosa(tick), että loppuaika saadaan vielä näkyviin kuluvan jakson näyttöön. Jos tietokantaoperaatio ei onnistu, nollataan var-, prms- ja hist–muuttujat ja suljetaan tietokantayhteys. Vtrin osaa piirtää palkit ja käyrät tietokannan historiataulusta aina voimassa olevan ajan mukaan.

Hetkellisarvoihin kirjoitetaan kaikille kolmelle rajalle ajat, jolloin ne ennusteen mukaan saavutetaan. Lisäksi hetkellisarvoon kirjoitetaan näille kolmelle rajalle tiedot siitä, paljonko teho saa olla enintään loppujakson ajan, että pysytään vielä rajojen sisällä. Tietokantaan kirjoitetaan ykköshälytysrajan ylitys (overshoot), joka kertoo ennusteen siitä, paljonko raja tullaan ylittämään tai paljonko siitä jäädään alle. Lisäksi hetkellisarvoihin kirjataan jakson keskiarvokulutus (cycle average power), arvioitu kulutus (estimated power) sekä aika sekunteina jakson loppuun.

```
if (mAVIDNTable[i] > 0)
{
   time = start[i];
   mDriver.Variables.Request(AverageVarIDN
   var = mDriver.Variables[AverageVarIDN
   if (var == null)
   {
     var = null; LogMessage("Error while writing datas.");
     return false;
   }
```

```
    if (mDriver.CanFastCommitCurrentValues)
    {
      mDriver.FastCommitCurrentValues(var.Id,
(object)mPeriodAveragePowerTable[i], mDriver.ServerTimeInUTC,
ABB.Vtrin.cValueStatus.OK);
    }
}
```

Koodiesimerkki 4: Hetkellisarvoon kirjoitus

Koodiesimerkissä 4 nähdään, miten kirjoitetaan keskiarvotieto hetkellisarvoon. Ensin testataan, onko olemassa keskiarvotietoa. Sitten yhdistetään ajuriin (mDriver). Jos oikea taulu löytyy, tallennetaan hetkellisarvoon komennolla FastCommitCurrentValue ja parametreina annetaan muuttujan tunnusnumero (id), keskiarvo, palvelimen aika UTC-muodossa ja muuttujan status, joka tässä tapauksessa aina on OK.

3.2.3 Aikojen käsittely

Kantaan lukemista ja kirjoittamista varten tarvitaan jakson alku- ja päättymisajat. Alkuajan saaminen on kuitenkin hankalampaa kuin voisi kuvitella, koska halutaan, että sovellus toimii kaikilla aikavyöhykkeillä. Käyttöliittymässä pitää näkyä aina paikallinen aika, mutta ohjelma suorittaa laskut UTC-ajalla (koordinoitu yleisaika, Coordinated Universal Time). Ajat lasketaan calculateTimes-metodin sisällä.

Alkuajan saamiseksi tehtiin oma metodi, joka laskee alkuajan paikallisessa ajassa, käyttäen mModDiv-apumetodia. Ajan laskeminen tapahtuu eri tavalla riippuen siitä, onko ajalla kuukausikomponenttia vai ei. Ajalla on kuukausikomponentti, jos jakson pituudeksi annettu parametri on negatiivinen, eli kuukausia. Tämän jälkeen aika muunnetaan UTC-ajaksi.

Historiatietojen päättymisaika saadaan pyytämällä nykyinen UTC-aika. Jakson päättymisajan saa lisäämällä alkuaikaan jakson pituus joko kuukausina tai minuutteina sen mukaan, kuinka se on alun perin annettu. Suomessa on käytössä Itä-Euroopan aika (Eastern European Time, EET), joka on talvisin 2 tuntia ja kesäisin 3 tuntia UTC-aikaa edellä.[16.]

Datetime esittää ajanhetkeä eli päivämäärää ja kellonaikaa. [17.]

```
DateTime esimPaiva = new DateTime(2011, 9, 22, 15, 12, 0);
```

Koodiesimerkki 5: DataTimen luonti

Koodiesimerkissä 5 luodaan ajanhetki 15:12:00 22.9.2011. Parametreiksi annetaan vuosiluku, kuukausi, kuukauden päivä, tunnit, minuutit ja sekunnit. Parametreja voi myös olla enemmän tai vähemmän riippuen siitä, kuinka tarkka aika halutaan antaa.

TimeSpan taas kuvaa aikaeroa. Alkuperäisessä TimeSpanissa on parametreina vain päivät, tunnit, minuutit, sekunnit ja millisekunnit (sekunnin tuhannesosa). Kuitenkin tarvitsemme tehonvalvontasovelluksessa myös kuukausien käsittelyä. Huomattavaa on, että tässä yhteydessä ei käy kuukauden pyöristäminen 30 päiväksi, vaan halutaan tarkat kuukaudet. Tätä varten otamme käyttöön ABB Oy:n sisäisen laajennuksen cAdvancedTimeSpanin, joka laajentaa alkuperäistä System.TimeSpan-luokkaa muun muassa lisäämällä siihen ominaisuuden, joka mahdollistaa kuukausien käytön aikavälissä. [17.]

```
ABB.Vtrin.cAdvancedTimeSpan ValueInterval = new
ABB.Vtrin.cAdvancedTimeSpan(0, new TimeSpan(0, 0,
intervalSeconds);
```

Koodiesimerkki 6: AdvancedTimeSpanin toiminta

Koodiesimerkissä 6 näemme, miten luodaan AdvancedTimeSpan muuttuja, jonka aikaväli on yhden mittausvälin pituus. Kuukausikomponentti on tässä esimerkissä 0 ja luodaan AdvancedTimeSpanin sisälle System.TimeSpan, jonka pituudeksi tulee laskentavälin pituus.

3.3 Historia

Tarkkailtava historia lasketaan kumulatiivisesti eli kasautuvasti. Aloitetaan jakamalla hetkellisarvo laskentavälien kokonaislukumäärällä. Laskentavälien kokonaislukumäärän saa, kun jakaa ensin jaksonvälin pituuden sekunteina 60:llä. Seuraava palkki on siis aina edellinen palkki plus lisäys. Koska sovelluksessa tarvittiin sekä tarkkailtavan kumulatiivisia arvoja, jossa oli laskettu hetkellisarvot yhteen, että samaa arvoa jaettuna laskentavälien kokonaislukumäärällä, käytämme kahta eri metodia: mCumulativeCalcia ja mCumulativeCalcDivisoria. Näiden ainoa ero on se, että toisin kuin ensin mainitussa

metodissa, jälkimmäisessä kumulatiiviset arvot jaetaan laskentavälien kokonaislukumäärällä. [18.]

Huomattavaa on, että lisäys voi olla myös negatiivinen, jos hetkellisarvo on negatiivinen. mCumulativeCalcDivisor-metodin laskemien arvojen perusteella piirretään kuvaajaan historiapalkit, joista voidaan siis seurata tehonkulutuksen kertymistä jakson aikana. Kumulatiivisten arvojen perusteella lasketaan myös ennusteet.

3.4 Ennustaminen

Ennustamisen tarkoituksena on arvioida tehonkulutus mahdollisimman tarkasti, jotta yritys pystyy arvioimaan energiantarpeensa ja hankkimaan juuri oikea määrän sähköä ja näin saamaan sähkönsä mahdollisimman halvalla. Ennuste lasketaan koko jakson loppuun kuvaajana ja se lasketaan heti, kun historiataulussa kyseisessä jaksossa on yksikin arvo.

Jos käyttäjä ei ole antanut ennustetaulua, ennuste lasketaan viimeisimpien arvojen erotuksien keskiarvojen perusteella. Käytettävien historiatietojen määrä määräytyy sen mukaan, mitä käyttäjä on antanut kohdassa ExtrapolationIntervals. Jos käyttäjä on jättänyt ExtrapolationIntervals-kentän tyhjäksi, on oletusarvo 5 edellistä historiatietoa. Sovelluksessa on kuitenkin otettu huomioon, että jos historiatietoja on alle viisi, historiatietoja voidaan ottaa enintään se määrä, mikä historiatietoja on saatu kyseisellä jaksolla. Kumulatiivisesta kasvusta otetaan keskiarvo ja tämän perusteella tulevaisuuden arvoja kasvatetaan ennusteeseen jakson loppuun asti. Tällä tavoin saatava ennuste on hyvä silloin, jos hetkellisarvot ovat tasaisia. Kuitenkin jos arvot vaihtelevat paljon on ennuste melko epätarkka, koska tällöin myös ennuste pomppii, kun arvot, joista keskiarvo lasketaan, vaihtelevat suuresti.

Jos käyttäjä on antanut ennustetaulun, luetaan tässä kohdassa tietokannan ennustetaulusta tulevaisuuden ennusteet, skaalataan ne oikean suuruisiksi ja sijoitetaan ne sovelluksen tauluun, jossa on ennustetiedot. Nämä tallennetaan tietokantaan samaan tapaan kuin lasketutkin rivit. Tällainen etukäteen arvattu ennuste voi perustua esimerkiksi aiempiin kokemuksiin ja sijoitetaan esimerkiksi aiemman viikon tiedot ennusteeseen, jos arvot ovat helposti ennustettavissa.

3.5 Raja-arvojen ja jakson laskutoimitukset

Sovelluksessa on mahdollista antaa 3 eri raja-arvoa. Näille raja-arvoille suoritetaan laskutoimituksia. Raja-arvot asetetaan Vtrin-ohjelmassa painamalla hiiren oikeaa painiketta valitun rajan päällä ja valitsemalla Send to-> Status Display. Rajan voi asettaa auenneen ikkunan vasemmassa laidassa olevasta liukupalkista tai kirjoittamalla haluttu raja oikeanpuoleiseen tekstikenttään. Kaikille raja-arvoille lasketaan, kuinka kauan nykyisellä kulutuksella kestää saavuttaa jaksolle määritetty raja-arvo sekä mikä pitäisi kulutuksen olla loppujakson ajan maksimissaan, jotta rajaa ei ylitetä.

Sekuntirajat ja tehorajat lasketaan käyttäen apuna valuesToLimit-metodia.

```
mSecondsToLimit2 = mValuesToLimit(MonitLimit_2, EnergySum,
mEstimatedPower, PastIntervalsOfPeriod,
(long)(NofIntervalsInPeriod * ((double)IntervalSeconds / 60)),
(int)mSecondsToPeriodEnd).First;

mPowerToLimit2 = mValuesToLimit(MonitLimit_2, EnergySum,
mEstimatedPower, PastIntervalsOfPeriod,
(long)(NofIntervalsInPeriod * ((double)IntervalSeconds / 60)),
(int)mSecondsToPeriodEnd).Second;
```

Koodiesimerkki 7: Rajan 2 laskut

Koodiesimerkissä 7 lasketaan rajalle kaksi sekunti- ja tehorajat käyttämällä apuna mValuesToLimit-metodia. Laskettaessa sekuntirajaa metodille annetaan parametrina laskettavan rajan arvo, energiasumma, ennustus tehon arvoksi jakson lopussa, kuluneet laskentavälit jaksossa, laskentavälien lukumäärä, sekä tieto kuinka monta sekuntia on jakson loppuun. Tehorajan parametrit ovat muuten samat, paitsi ennustetaulun viimeisen arvon tilalle tulee arvio (estimatedPower). Metodi palauttaa parin (pair), jonka ensimmäinen muuttuja on sekuntiraja ja toinen voimaraja.

Lisäksi ykkösrajalle lasketaan ylitys (overshoot), joka kertoo, kuinka paljon ennusteen mukaan raja-arvo tullaan ylittämään. Ykkösrajan raja-arvon alittaminen merkataan negatiivisena ylityksenä eli miinusmerkkisenä numerona. Ylityksen laskeminen on erityisen tärkeää kulutuksen seuraamisen kannalta. Esimerkiksi ulkopuolisen ohjelman

voi laittaa seuraamaan ylitystä ja tekemään hälytyksen, jos ylitys poikkeaa nollasta. Ylityksen yksikkönä on Megawatti (MW).

Jaksolle lasketaan jakson keskiarvokulutus (cycle average power), arvioitu kulutus (estimated power) ja aika sekunteina jakson loppuun. Keskiarvokulutus saadaan laskemalla historiatietojen keskiarvo. Arvioitu kulutus saadaan laskemalla määritellyn ajan (oletuksena 5 väliä) historiatietojen kasvusta keskiarvo ja laskemalla kulutus jakson lopussa, jos koko ajan kasvu jatkuisi tämän keskiarvon mukaan.

Monet edellisistä laskutoimituksista on vanhassa koodissa tehty antaen eri parametreja CSCommon_Time -luokan SubAndDivide-metodille. Kuitenkaan sovelluksessa ei ollut mahdollista kutsua kyseistä C++-luokan metodia, joten päädyttiin kääntämään SubAndDivide metodi C#-ohjelmointikielelle ja liittämään se osaksi tehonvalvontasovelluksen koodia cTieLineMonitor-luokan metodiksi.

Metodille annetaan kaksi DateTime-tyypin ja yksi ABB.Vtrin.cAdvancedTimeSpan-tyypin parametri. Metodi palauttaa pair-olion, joka sisältää long ja cAdvancedTimeSpan tyyppiset muuttujat. C++-ohjelmointikielen <utility>-header-tiedostossa on määritelty geneerinen pair luokka (template). Pair eli pari on muuttujatyyppi, mihin saa sisällytetyksi kaksi muuttujaa, joita kutsutaan tyyliin pari.First tai pari.Second. Muuttujat voivat olla mitä tyyppiä tahansa, sen mukaan miten ne on määritelty. Muuttujien ei myöskään tarvitse olla samaa tyyppiä keskenään. C#-ohjelmointikielessä vastaavaa muuttujatyyppiä ei kuitenkaan ole, joten jouduttiin sisällyttämään koodiin myös käsintehty pair-luokka.[19.]

```
public class pair<T1, T2>
  {
   public T1 First
   {
    get { return this.t1; }
    set { this.t1 = value; }
   }

   public T2 Second
   {
    get { return this.t2; }
    set { this.t2 = value; }
   }

   public pair(T1 type1, T2 type2)
```

```
    {
     this.t1 = type1;
     this.t2 = type2;
    }
    private T1 t1;
    private T2 t2;
  }
```

Koodiesimerkki 8: pair-luokan luonti[19.]

Koodiesimerkissä 8 näemme, miten C#-ohjelmointikielessä saa luotua C++-luokan paria vastaavan muuttujatyypin, jota voi siis käyttää samaan tapaan kuin C++:n paria. T1 ja T2 ovat muuttujien tyypit, ja ne voivat olla mitä hyvänsä sen mukaan, miten ne on määritelty. Luodaan set- ja get- metodit First- ja Second-muuttujille ja alustetaan ne. Pari on tärkeä ja usein käytetty luokka, kun käännetään ohjelmia C++-ohjelmointikieleltä C#-ohjelmointikielelle. Niinpä monet muutkin ohjelmoijat ovat törmänneet ja tulevat törmäämään samaan ongelmaan ja kyseinen pair-luokka löytyykin netistä valmiina.

mSubAndDivide saa parin First-arvoksi sen, kuinka monta kertaa annettu aikaväli mahtuu alku- ja loppuajan väliin. Ensimmäisen aikaparametrin tulee aina olla myöhäisempi kuin jälkimmäinen. Tämän avulla saamme selville, kuinka monta laskuväliä, minuuttia ja sekuntia on kulunut jaksosta. mSubAndDivide-metodin avulla saamme myös laskettua kuluneiden sekuntien määrän laskentavälissä ja kuinka monta laskentaväliä on jaksossa. Metodin toista arvoa (Second) ei tämänhetkisessä sovelluksessa tarvita.

3.6 Lokitulostus

Sovelluksen ohessa pidetään myös lokia, johon kirjataan ohjelman suoritusten kannalta keskeisiä asioita, kuten sattuneet virheet. Näin käyttäjä näkee heti, missä on sattunut virhe ja mistä se johtuu. Virheellisten syötteiden sattuessa ohjelman ei tule kaatua, vaan se kirjoittaa virheen syyn lokiin ja hyppää muuten suorituksessa kyseisen kohdan yli.

Esimerkiksi aina tietokantaoperaatioita suorittaessa kirjataan lokiin, onnistuiko operaatio, koska tietokantaoperaatiot ovat usein alttiita virheille. Myös onnistuneet

sovelluksen suorittamiset kirjataan lokiin. Koska jokaiselle tunnille luodaan aina oma loki, niin niitä on helppo tarkastella jälkikäteen.

4 Uudet ominaisuudet

Sovelluksen kääntämisen yhteydessä oli myös otollinen aika tehdä sovellukseen lisäyksiä, joita on huomattu tarpeellisiksi, kun sovellus on ollut asiakaskäytössä. Nämä myös vaativat parametrien lisäämistä tietokantaan. Muutoksia tietokantaan ei ole vielä toteutettu, joten tällä hetkellä ominaisuuksia testatakseen pitää haluamansa arvot kovakoodata koodiin. Muutokset tietokantaan ja käyttöliittymän ovat kuitenkin tulossa, ja uusissa ominaisuuksien käytöstä puhutaan niin kuin ne tulevat olemaan.

4.1 Pienimmän neliösumman menetelmä

Tehonvalvontasovelluksessa on käytössä laskutapa, jossa historiapylväät lasketaan kumulatiivisesti laskien yhteen hetkellisarvot jaettuna arvojen kokonaismäärällä. Tämä aiheuttaa sen, että jos hetkellisarvot vaihtelevat suuresti historiapylväät ovat epätasaisia. Tämän perinteisen tavan rinnalle luotiin uutena ominaisuutena lasku pienimmän neliösumman menetelmällä, jonka tarkoituksena on näyttää tasaisia historiapylväitä, sekä pyrkiä mahdollisimman tarkkaan ennustukseen. Käyttäjä valitsee itse, kumpaa tapaa haluaa käyttää. Tällöin myös mukaan otetaan koko aikaväli, eikä vain muutamaa uusinta hetkellisarvoa, joten ennustukset eivät hypi jakson kuluessa yhtä rajusti kuin laskettaessa kumulatiivisesti.

Pienimmän neliösumman menetelmä on peruslähestymistapa matemaattisessa optimoinnissa, kun halutaan saada mitatulle aineistolle paras mahdollinen sovite. Ideana pienimmän neliösumman menetelmässä on minimoida sovitteen ja aineiston pisteiden välisten erotusten eli residuaalien neliösummaa. Tärkein sovellutus pienimmän neliösumman menetelmälle on suoran sovitus. Pienimmän neliösumman ongelmat voidaan jakaa lineaarisiin ja epälineaarisiin. Pienimmän neliösumman menetelmässä estimoidaan kertoimet (kulmakerron ja vakio), jotka antavat pisteparven läpi kulkevan suoran, joka kertoo mallissa olevien selitettävän muuttujan (y) ja selittävien muuttujien (x) välisen riippuvuuden. [20; 21.]

Saksalainen matemaatikko Carl Friedrich Gauss käytti ensimmäisenä pienimmän neliösumman menetelmää vuonna 1794. Ensimmäisenä kirjan julkisti hänestä riippumatta ranskalainen Adrien-Marie Legendre. Sen jälkeen menetelmästä on tullut tärkeä väline astronomeille ja geodeetikkoille. Yhteiskuntatieteisiin pienimmän neliösumman menetelmä tuotiin 1900-luvun alussa, ja se on vielä nykyäänkin kaikista tärkein väline, kun kaoottiselta näyttävään kahden tai useamman muuttujan arvoista muodostuvaan pisteparveen halutaan saada järjestystä. [21.]

Käyttäjä valitsee käyttöliittymässä, haluaako käyttää laskuissa perinteitä kumulatiivista laskutapaa vai uutta pienimmän neliösumman menetelmää. Lähtökohtaisesti käytetään vielä toistaiseksi vanhaa tapaa. Uutta tapaa ei ole vielä testattu käytössä ja siitä vasta kerätään testituloksia ja tutkitaan, missä tilanteissa se olisi mahdollisesti parempi kuin vanha. Uusi tapa reagoi muutoksiin paljon hitaammin kuin vanha, koska ottaa huomioon koko laskuvälin.

Suoritetuissa testeissä ilmeni, että jos ennustetaan PC:n prosessorin kuormitusta, pienimmän neliösumman menetelmä on parempi tapa kuin vanha, koska kuormitus voi hetkellisesti pompata usean laskentavälin ajan korkealle (kun prosessoria kuormitetaan) ja palata taas takaisin normaalille tasolle. Tällöin vanhalla tavalla ennuste pomppaa ylös ja laskee vähän ajan päästä taas alas. Uusi tapa taas tekee tasaisempaa ennustetta, koska se pohjautuu koko laskentajaksolle. Päätelmää voidaan soveltaa niin, että vanha tapa on parempi kuin on tärkeää reagoida nopeasti ja uusi tapa, kun halutaan jättää äkilliset vaihtelut huomioimatta.

Aluksi ohjelma toimii molemmilla laskutavoilla samalla lailla. Ennustusvaiheessa kuitenkin, jos pienimmän neliösumman menetelmä on valittu, sovellus hyppää mLeastSquares metodin sisään. Parametrikseen metodi saa taulukon, jossa on historiatietojen hetkellisarvot. Kyseinen taulukko annetaan pääluokan parametrina mForecast-metodille.

```
System.Collections.Generic.List<APP_LSF.Point> points = new
List<APP_LSF.Point>();
for (int i = 0; i < cDataBaseValues.Length; i++)
{
```

```
   //Use Polynomial Least Squares Fit function to get power
value for speed
   points.Add(new APP_LSF.Point(i + 1, cDataBaseValues[i]));
}
spcpowerxx = APP_LSF.FindPolynomialLeastSquaresFit(speed,
points, 1);
```

Koodiesimerkki 9: Pienimmän neliösumman sovite

Koodiesimerkissä 9 nähdään, kuinka pistejoukko sovitetaan pienimmän neliösumman menetelmällä yhdelle suoralle. Apuna käytetään APP_LSF-apuluokkaa, joka on tehty pienimmän neliösumman laskemiseen. Kyseinen luokka soveltuu myös korkeampien asteiden suorien laskemisiin, mutta tehonvalvontasovelluksessa tarvitaan vain suoraa, eli lineaarista käyrää.

Aluksi luodaan lista, johon tulee APP_LSF.Point-tyyppisiä pisteitä. Sitten käydään läpi kaikki kumulatiiviset historia-arvot, ja tehdään näistä pisteitä, joiden x-arvo on sen järjestysnumero mDataBaseValues-taulussa ja y-arvo on kumulatiivinen historia-arvo. Tämän jälkeen pisteet liitetään listaan. Seuraavaksi lasketaan sovite (spcpowerxx) käyttäen APP_LSF-luokan FindPolynomialLeastSquaresFit-metodia, jolle annetaan parametreiksi speed, joka on määritetty ykköseksi, koska laskemme suoraa. Lisäksi annamme parametreiksi edellä laaditun pistelistan, sekä ykkösen taas, koska ensimmäisen asteen käyrä on kyseessä. Useamman asteen käyrille ei ole käyttöä tehonvalvontasovelluksessa.

```
mLeastSquaresTable = new double[cDataBaseValues.Length];
if (cDataBaseValues.Length == 1)
{
 mLeastSquaresTable[0] =
mCumulativeCalcDivisor(cDataBaseValues)[0];
}
else if (cDataBaseValues.Length <= 0)
{
 mLeastSquaresTable[0] = 0;
}
else
{
 for (int i = 0; i < cDataBaseValues.Length; i++)
 {
  mLeastSquaresTable[i] = (spcpowerxx / mTicsNumber) * (i + 1);
 }
}
```

Koodiesimerkki 10: Historian sovitus suoraan

Koodiesimerkistä 10 käy ilmi, miten saadun sovitteen avulla saadaan sovitettua historiatietojen pisteet suoraksi. Olion sisäisen pienimmän neliösumman sovitetaulun pituudeksi asetetaan sama, kuin kumulatiivisten historiatietojen pituus on. Jos historia-arvoja on vain yksi, arvo on kuitenkin suoraan kyseinen historia-arvo. Jos taas arvoja ei ole, annetaan ainoaksi arvoksi tauluun 0. Muussa tapauksessa tehdään jokaiselle historia-arvolle sovitus suoralle ja sijoitetaan arvot olion sisäiseen tauluun. Globaalia mLeastSquaresTable–taulua käytetään myöhemmin kirjoittaessa historiatietoja tietokantaan.

Lopuksi laaditaan vielä samalla sovite ennusteelle, joka noudattaa samaa suoraa kuin historiakin. Suoraa siis jatketaan jakson loppuun asti samalla kulmakertoimella ja lasketut arvot sijoitetaan cForecastSquares-tauluun. Lopuksi mLeastSquares-metodi palauttaa edellä mainitun taulun, joka sitten asetetaan ennustusmetodissa kyseisen rivin ennustetauluksi, josta ennusteet kirjataan tietokantaan.

Pienimmän neliösumman menetelmän lisäys lasku- ja ennustustavaksi aiheutti myös tietokantaan muutoksia. Kun tietokannassa tallennetaan tiedot, avataan ensin tietokantayhteys normaalisti. Kuitenkin nyt tallennetaan historia-arvoihin sovitetut arvot mLeastSquaresTable taulusta. Tulevaisuusarvot tallennetaan taas futureData-taulusta aivan kuin kumulatiivisienkin arvojen kohdalla.

4.2 Hälytysrajat

Jo alkuperäisessä sovelluksessa laskettiin kulutukselle ylitys. Jos ylitys tapahtuu, voidaan haluta tuottaa hälytys ulkopuolisen ohjelman avulla, jotta tilanteeseen pystytään reagoimaan mahdollisimman nopeasti. Kuitenkaan kaikissa tilanteissa ei hälytyksen antaminen vastaa tarkoitusta. Siksi onkin hyödyllistä asettaa ylitykselle rajoja. Tällöin ylitys näyttää aina nollaa, jos pysytään sallittujen rajojen sisällä.

4.2.1 Hälytysrajat kuluneen ajan mukaan

Tehonvalvontasovellukseen lisättiin ominaisuus, että käyttäjä voi antaa parametrina, kuinka monta prosenttia ajasta on kulunut ennen, kun hälytys annetaan. Esimerkiksi jos jakson pituus on 60 minuuttia ja annetaan rajaksi 25 %, hälytys tulee vasta, jos 15

minuutin kohdalla ennustus näyttää, että tulisi ylitys. Jos aivan jakson alussa ennuste näyttää ylitystä, ei välttämättä ole vielä syytä huolestua. Aikarajan asettaminen on hyödyllistä etenkin, jos mittausten hetkellisarvot vaihtelevat paljon.

4.2.2 Hälytysrajat ylityksen mukaan

Lisäksi tehonvalvontasovellukseen lisättiin myös ominaisuus, että käyttäjä pystyy määrittelemään, kuinka suuri ylityksen pitää olla, että se raportoidaan. Jos ylitys on hyvin pieni, ei hälytystä kannata silloin tehdä. Hälytyksen ylitykselle määritellään raja antamalla parametrina, kuinka monen prosentin ylityksestä tulee hälytys. Esimerkiksi, jos annetaan parametriksi 100, hälytys tulee, jos ylitys on rajan suuruinen.

Pitkälti yrityksien solmimista energiasopimuksista on kiinni, kuinka suuri merkitys yritykselle on sillä, että se tilaa liian suuren energiamäärän. Usein kannattaa pyrkiä mahdollisimman tarkasti arvioimaan tehon ja energian käyttö. Siksi myös negatiivinen ylitys ilmoitetaan ja siitäkin voidaan tehdä hälytys. Sovellukseen lisättiin myös parametri negatiivisen ylityksen määrälle. Esimerkiksi laittamalla parametriksi 10 hälytys tulee, jos ei saavuteta edes 90 prosenttia rajasta. Negatiivinen ylitys voidaan jättää myös kokonaan huomioimatta antamalla parametrille arvoksi 100 %. Jokaiselle kolmelle rajalle määritellään siis erikseen ylärajat ja alarajat.

4.3 Valvontarajojen otto historiataulusta

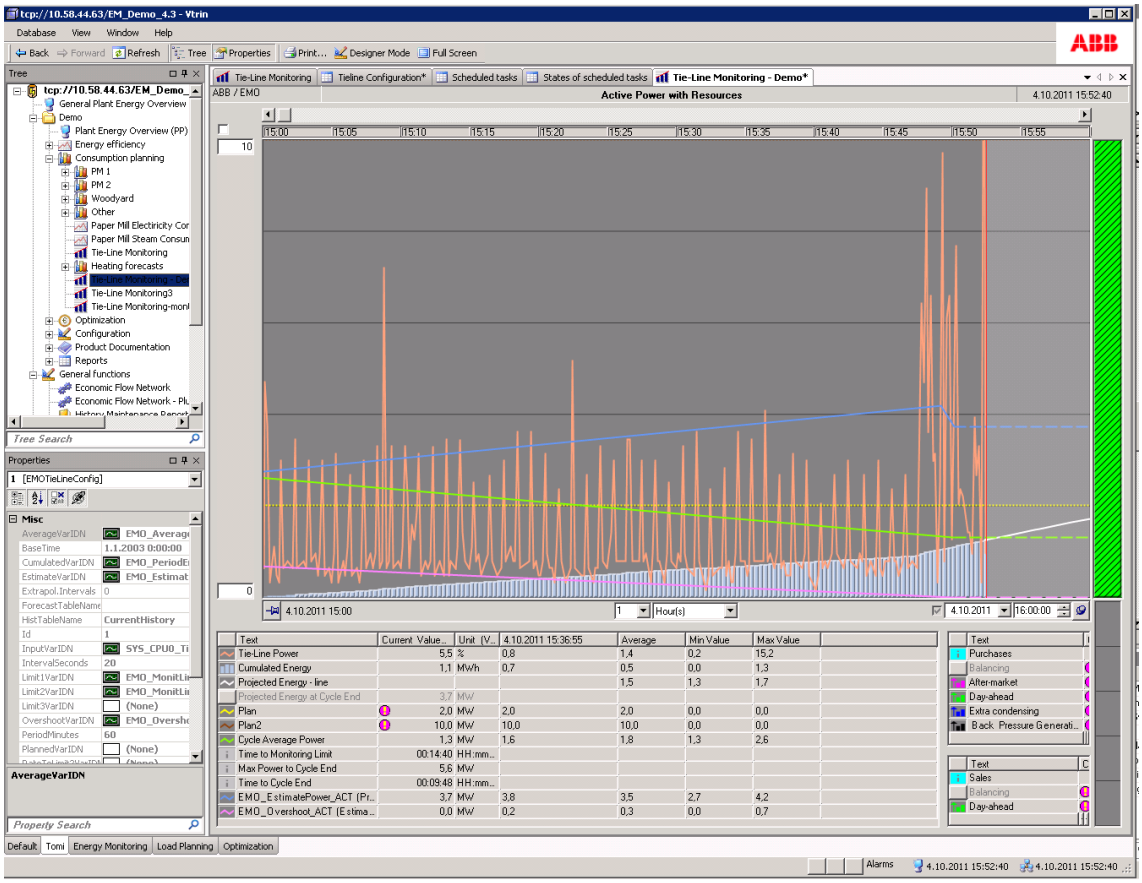
Yksi esiin tullut ja toteutettu uudistus oli valvontarajan ottaminen annetusta taulusta hetkellisarvon sijasta. Tätä varten määrittelytauluun lisättiin sarake LimitHistoryTable. Sovellus tarkastaa aina jokaisen rajan kohdalla, onko kyseisessä taulussa annettu arvo raja-arvolle. Jos taulua ei ole annettu, haetaan raja vanhaan tapaan hetkellisarvosta, mutta muuten otetaan arvo taulusta oikeasta kohtaan. Etuna tähän on, että rajat voivat esimerkiksi tiettyinä päivinä tai vuorokauden aikoina olla tietyt, jolloin ne saa tällä tavalla helposti määriteltyä etukäteen, kun ne muuten joutuisi aina manuaalisesti vaihtamaan uudelleen.

5 Testiajot

Testiajot suoritettiin suorittamalla ohjelma manuaalisesti ja tarkastelemalla sitten tuloksia käyttöliittymästä käsin. Kyseessä on sama näyttö, jota sovelluksen käyttäjä käyttää.

5.1 Testiajo 1: Kumulatiivinen laskutapa

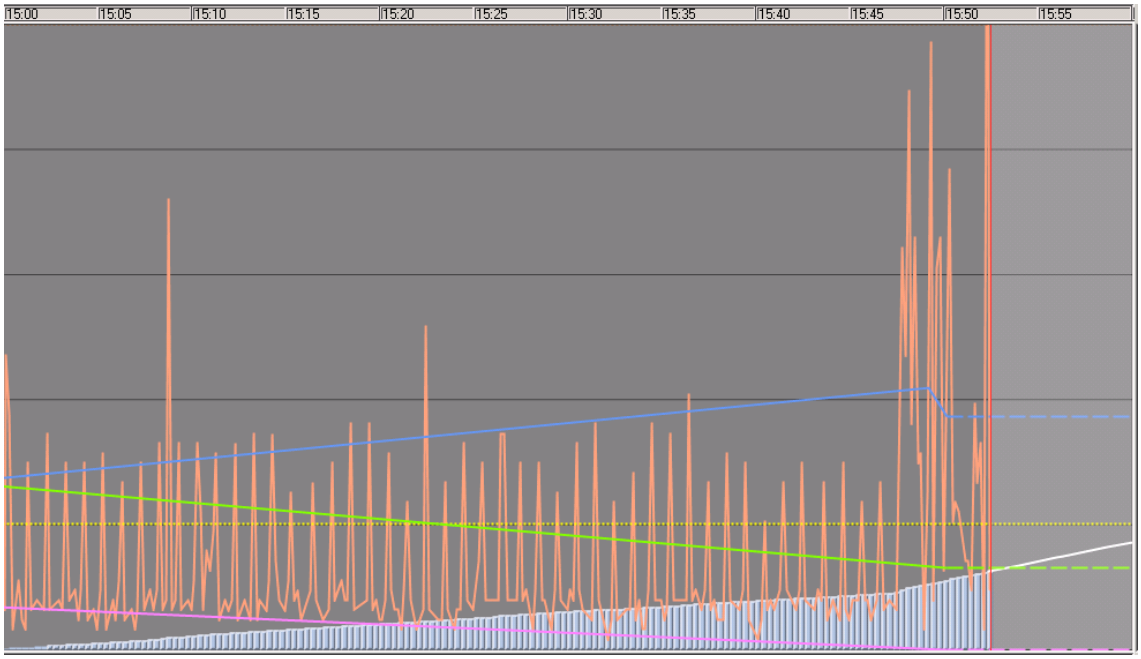
Testiajossa yksi otimme muuttujaksi CPU0-ajan, koska se vaihtelee paljon ja antaa hyvän kuvan tapauksesta, jossa teho on epätasaista. Mitattavan jakson pituus on yksi tunti ja mittausväli 20 sekuntia.



Kuva 7: Testiajon koko näkymä

Kuvassa 7 näemme testiajon 1 koko näkymän. Vasemmalla ylhäällä on määrittelypuu, josta kaavio valitaan. Vasemmalla alhaalla on konfigurointiapuikkuna. Ylhäällä on Vtrinin valintapalkki ja sen alapuolella välilehdissä auki olevat kaaviot. Keskellä on kuvaaja. Kuvaajan vasemmalta puolelta valitaan tarkkailtavalle arvolle ylä- ja alarajat.

Tässä tapauksessa alarajana 0 kW ja ylärajana 10 kW. Kuvaajan alapuolelta valitaan tarkkailtavan ajan pituus ja oikealta kuvaajan alapuolelta valitaan jakson päättymisajankohta. Alhaalla keskeltä näkyvät mitattavat muuttujat ja niiden hetkellisarvot. Oikealla alhaalla ja äärioikealla olevat tiedot eivät taas ole tehonvalvontasovelluksen kannalta merkityksellisiä, vaan niitä käytetään muissa energianhallintaohjelmiston sovelluksissa.



Kuva 8: Testiajon 1 kaavio

Kuvassa 8 näemme tarkemmin testiajon 1 kaavion. Tarkastelemme kuvaa pari minuuttia tehonvalvontasovelluksen suorittamisen jälkeen. Oranssi viiva kuvaa hetkellisarvoja. Esimerkissä hetkellisarvot vaihtelevat hyvin paljon ja laskentavälille mahtuu yksi näkymän rajan 10 kW ylitys hieman ennen nykyhetkeä. Keltainen viiva kuvaa asetettua hälytysrajaa. Violetti viiva kuvaa arvioitua hälytysrajan ylitystä. Testiajossa 1 näyttää ensin, että hälytysraja ylitetään, mutta koska myöhemmin jaksolla hetkellisarvot keskimäärin pienenevät, alkaa ylitys näyttää nollaa jakson loppupuolella. Testiajossa 1 hälytyksen alarajaprosentiksi on laitettu 100 %, eli rajan alle jäävää ylitystä ei mitata.

Vihreä viiva kuvaa hetkellisarvojen keskiarvoa jakson alusta kyseiseen hetkeen. Sininen viiva kuvaa arviota, eli tässä tapauksessa viiden edellisen arvon keskiarvoa. Siniset palkit kuvaavat tehon kasvua kumulatiivisesti. Kuvasta on huomattavissa sinisten palkkien epätasaisuus, joka johtuu hetkellisarvojen vaihtelusta. Valkoinen viiva kuvaa

tehon kumulatiivisen kasvun kehityksen ennustetta. Valkoisen viivan loppupiste on siis ennuste jakson tehonkulutukseksi. Käyttöliittymä jatkaa automaattisesti sinisten palkkien piirtämistä valkoisen ennustekäyrän mukaan. Myös hetkellisarvo piirretään nykyhetken mukaan. Laskukohdasta eteenpäin ylitys, keskiarvo ja arvio piirretään katkoviivalla ja ovat sama kuin viimeksi laskettu arvo.

Text	Current Value...	Unit (V...	4.10.2011 15:36:55	Average	Min Value	Max Value
Tie-Line Power	5,5	%	0,8	1,4	0,2	15,2
Cumulated Energy	1,1	MWh	0,7	0,5	0,0	1,3
Projected Energy - line				1,5	1,3	1,7
Projected Energy at Cycle End	3,7	MW				
Plan	2,0	MW	2,0	2,0	0,0	0,0
Plan2	10,0	MW	10,0	10,0	0,0	0,0
Cycle Average Power	1,3	MW	1,6	1,8	1,3	2,6
Time to Monitoring Limit	00:14:40	HH:mm...				
Max Power to Cycle End	5,6	MW				
Time to Cycle End	00:09:48	HH:mm...				
EMO_EstimatePower_ACT (Pr...	3,7	MW	3,8	3,5	2,7	4,2
EMO_Overshoot_ACT (Estima...	0,0	MW	0,2	0,3	0,0	0,7

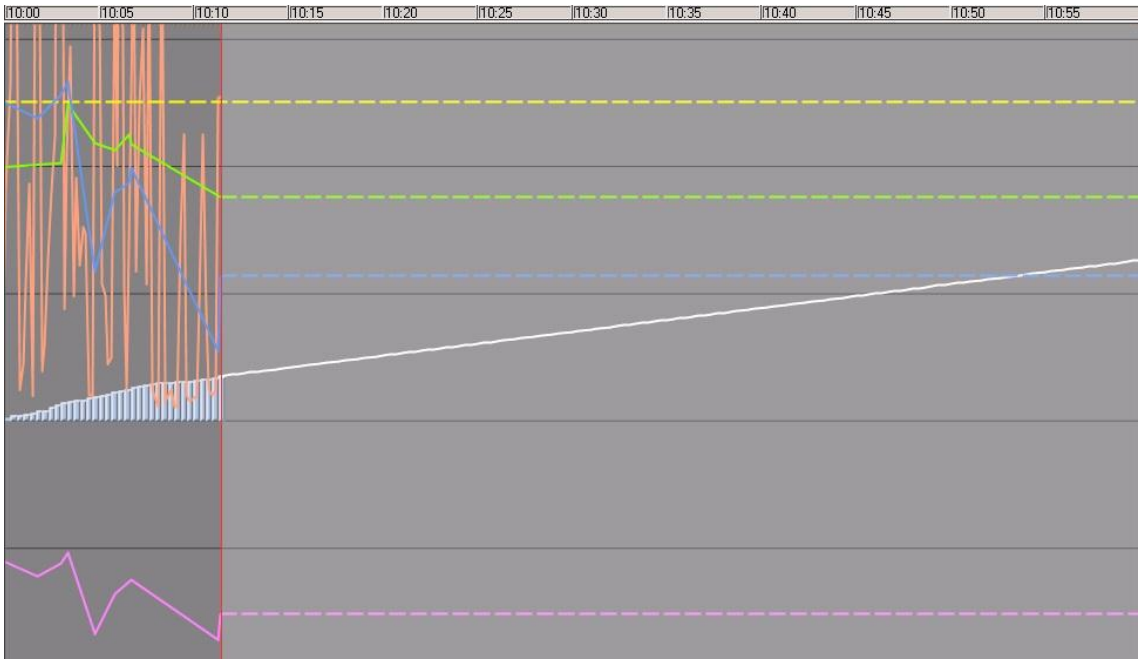
Kuva 9: Testiajon 1 muuttujat ja mitattavat hetkellisarvot

Kuvasta 9 käyvät ilmi mitattavat suureet, niiden hetkellisarvot, yksiköt, keskiarvot sekä minimi- ja maksimiarvot. Esimerkiksi näemme, että hetkellisarvot vaihtelevat 0,2 ja 15,2 välillä. Vaihtelu on siis huomattavaa. Cumulated Energy kertoo kulutetun kumulatiivisen tehon megawattitunteina siihen mennessä. Projected Energy at Cycle End kertoo ennusteen kulutukselle jakson lopussa. Kuten näemme kuvasta, hälytysraja 1 (plan) on 2 MW ja hälytysraja 2 (plan 2) 10 MW. Hälytysrajaa 3 ei ole annettu. Hetkellisarvojen keskiarvo laskentahetkellä on 1,3 MW. Ylimmällä rivillä Vtrinin laskeman keskiarvon ero sovelluksen laskemaan selittyy sillä, että ensin mainittu laskee keskiarvon nykyhetkeen asti ja jälkimmäinen laskentahetkeen.

Time to Monitor Limit- rivi näyttää, kuinka pitkä aika on ylitykseen, ja Time to Cycle End-sarake, kuinka pitkä aika on jakson loppuun. Tässä tapauksessa kun ylitykseen on 14 minuuttia ja jakson loppuun yhdeksän minuuttia, pysytään siis rajan sisällä koko ajan. Max Power to Cycle End -rivi näyttää, että loppujakson ajan voidaan kuluttaa tehoa keskimäärin 5,6 MW ja silti tullaan pysymään hälytysrajan sisällä. Viidestä edellisestä historia-arvosta laskettu arvio näyttää 3,7 MW. Tässä esimerkissä alitusta ei mitata, joten ylitys näyttää nollaa. Pientä ylitystä ei mitata ja ylitysrajaksi on määritelty 20 %.

5.2 Testiajo 2: Alitus

Testiajossa 2 poistettiin hälytysrajat niin, että ylitys tai alitus näytetään kaikissa tilanteissa. Hälytysraja valittiin tarkoituksella niin, että siitä jäädään jälkeen. Jakson aikana testiajossa 2 käytettiin 60 minuuttia ja laskentavälinä 20 sekuntia. Mitattavana muuttuja oli edellisen kohdan tapaan SYS_CPU0_Time. Näytön ylärajana pidettiin 10 MW, mutta alarajaksi muutettiin -6 MW, jotta myös alituskäyrä näkyisi kuvassa. Testiajossa 2 käytetään testiajon 1 tapaan kumulatiivista laskutapaa.



Kuva 10: Alituksen kuvaaja

Kuvassa 10 näemme kuvaajan tilanteesta, jossa syntyy alitus. Nollaviiva on kohdassa, josta sinisiä palkkeja aletaan piirtää. Kuten näemme kuvasta, alitetaan reilusti hälytysraja, koska violetti viiva on kokoajan sinisten palkkien alapuolella. Tällöin on hankittu selvästi liikaa energiaa. Huomaamme myös, että ylityksen vaihtelu on hyvin voimakasta aluksi. Tästä syystä ei yleensä heti aluksi kannata tehdä hälytystä.

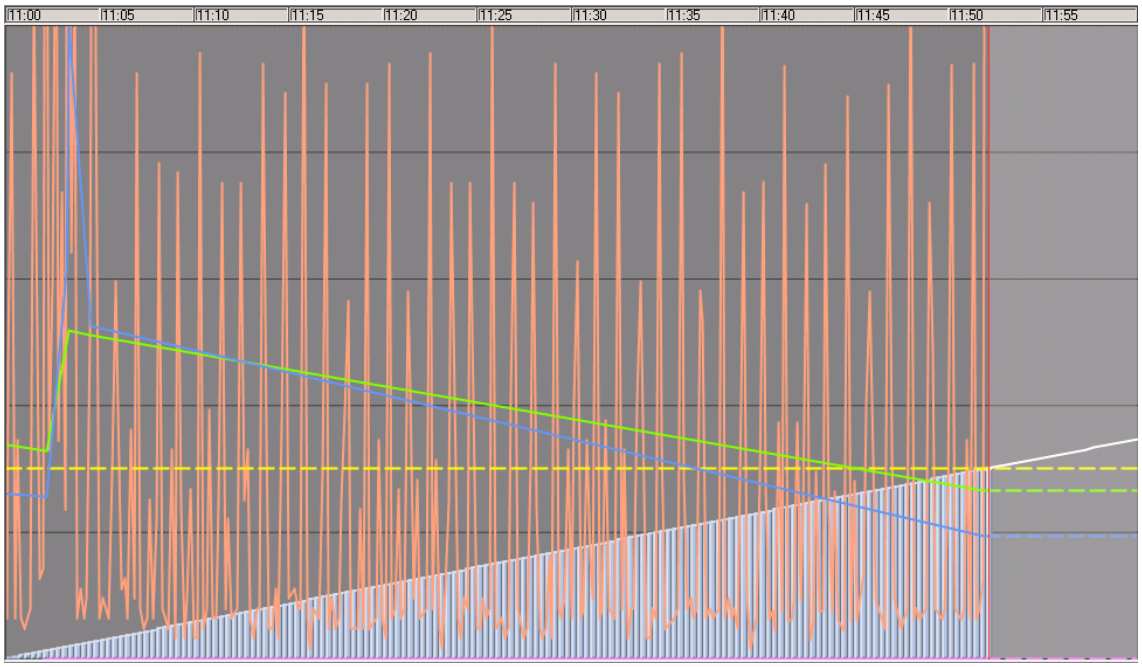
Text	Current Value...	Unit (V...	5.10.2011 10:04:01	Average	Min Value	Max Value
Tie-Line Power	17,5	%	4,4	5,6	0,3	24,9
Cumulated Energy	1,1	MWh	0,5	0,7	0,1	1,1
Projected Energy - line				2,6	1,1	4,0
Projected Energy at Cycle End	3,6	MW				
Plan	8,0	MW	8,0	8,0	8,0	8,0
Plan2	10,0	MW	10,0	10,0	0,0	0,0
Cycle Average Power	5,6	MW	7,5	5,8	5,6	8,0
Time to Monitoring Limit	01:54:45	HH:mm...				
Max Power to Cycle End	8,6	MW				
Time to Cycle End	00:48:35	HH:mm...				
EMO_EstimatePower_ACT (Pr...	3,6	MW	6,2	4,0	1,7	8,5
EMO_Overshoot_ACT (Estima...	-4,8	MW	-4,3	-4,8	-5,5	-3,3

Kuva 11: Testiajon 2 muuttujat ja mitattavat hetkellisarvot

Kuvasta 11 näemme mitattavat suureet, niiden hetkellisarvot, yksiköt, keskiarvot, sekä minimi- ja maksimiarvot. Esimerkiksi näemme, että hetkellisarvot vaihtelevat 0,3 ja 24,9 välillä, joten vaihtelu on ollut todella huomattavaa. Energian kulutusta mittaushetkellä on kertynyt 1,1 MWh. Kuvasta nähdään, että hälytysraja on 8 MW. Keskiarvoinen tehonkulutus on 5,6 MW. Nykyisellä tahdilla rajan saavuttamiseen menisi tunti ja 54 minuuttia ja jakson loppuun on enää 48 minuuttia, siis tehoa jää käytettäväksi vielä runsaasti seuraavan jakson puolellekin. Ylitys on ollut mittaushetkellä -4,8 MW, eli toisin sanoen alitus on 4,8 MW. Alitus on vaihdellut 5,5 MW ja 3,3 MW välillä.

5.3 Testiajo 3: Pienimmän neliösumman menetelmä

Testiajossa 3 käytetään uutta pienimmän neliösumman menetelmää. Jakson pituus on yksi tunti ja laskentaväli 60 sekuntia. Mitattava muuttuja on sama SYS_CPU0_Time kuin aikaisemmissa kuvissa. Näytön tarkkailuväliksi palautimme 0 MW – 10 MW, koska testiajon 3 kuvastamassa tilanteessa ei ole tarpeen seurata alitusta.



Kuva 12: Pienimmän neliösumman menetelmä

Kuvassa 12 näkyy kaavio, jossa pylväät on sovitettu pienimmän neliösumman sovitteen mukaan. Kuten näemme, pylväät ovat nyt tasaisesti nousevia toisin kuin kumulatiivisesti laskettaessa. Ylityksen rajoiksi on asetettu 20 % pituudella, 100 % alitukselle ja 10 % ylitykselle. Myös ennustekäyrä on sovitettu samalle suoralle historiapylväiden kanssa. Kuvasta myös huomaamme, että pienimmän neliösumman menetelmällä laskettu ennuste voi erota paljonkin keskiarvosta (vihreä viiva) ja viiden edellisen arvon keskiarvon perusteella lasketusta arviosta (sininen viiva). Tähän syynä on, että keskiarvosuoran tehtävä ei ole näyttää ennustetta, vain pelkästään hetkellisarvojen keskiarvoja ja sininen viiva taas kuvastaa vanhaa tapaa, jossa ennuste vaihtelee paljon.

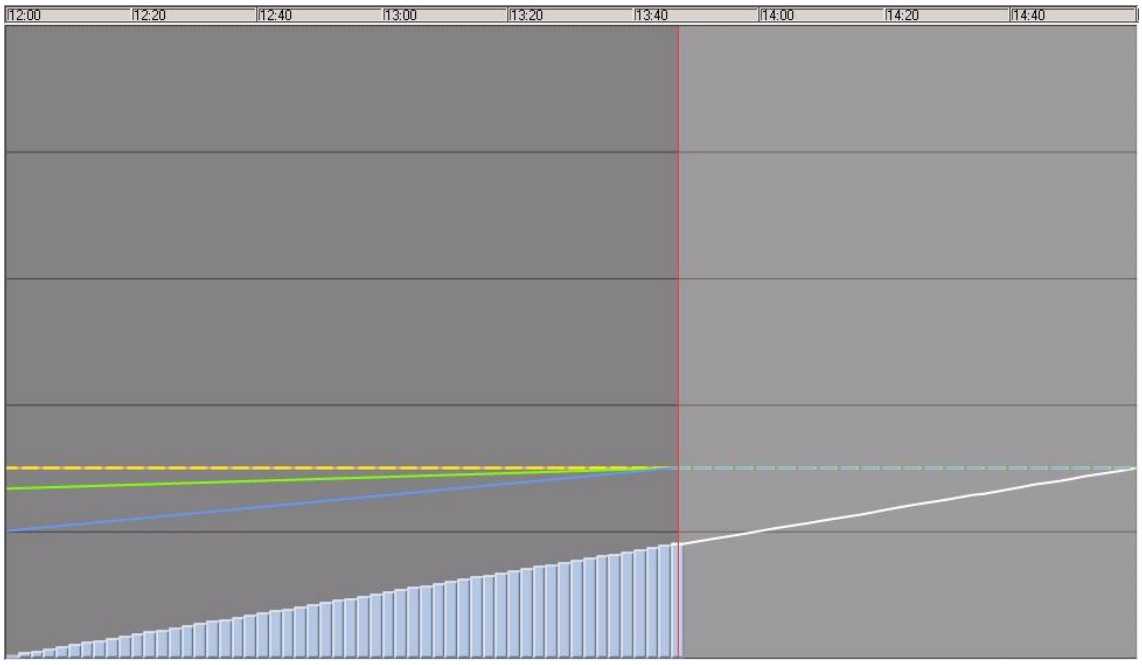
Text	Current Value...	Unit (V...	5.10.2011 11:04:24	Average	Min Value	Max Value
Tie-Line Power	3,4	%		2,7	0,2	23,3
Cumulated Energy	2,3	MWh	0	1,5	0,0	3,0
Projected Energy - line			0,3	3,2	3,0	3,5
Projected Energy at Cycle End	1,9	MW				
Plan	3,0	MW	3,0	3,0	3,0	3,0
Plan2	10,0	MW	10,0	10,0	0,0	0,0
Cycle Average Power	2,7	MW	3,3	3,7	2,7	5,2
Time to Monitoring Limit	00:12:20	HH:mm...				
Max Power to Cycle End	5,3	MW				
Time to Cycle End	00:08:07	HH:mm...				
EMO_EstimatePower_ACT (Pr...	1,9	MW	2,6	3,4	1,9	10,0
EMO_Overshoot_ACT (Estima...	0,0	MW	0,0	0,0	-0,2	0,0

Kuva 13: Testiajon 4 muuttujat ja mitattavat hetkellisarvot

Kuvassa 13 näemme mitattavat suureet, niiden hetkellisarvot, yksiköt, keskiarvot sekä minimi- ja maksimiarvot. Esimerkiksi näemme hetkellisarvojen vaihdelleen kuluneen jakson aikana 0,2 MW ja 23,3 MW välillä. Hälytysraja on 3 MW. Tehonkulutuksen keskiarvo on 2,7 MW. Aikaa kuluisi nykytahdilla hälytysrajan saavuttamiseen 12 minuuttia, mutta jakson loppuun on aikaa enää 8 minuuttia. Ylitys jää nollaksi, koska hälytyksen alarajaksi on määritelty 100 % eli sitä ei huomioida. Nykytahdilla kulutus saisi olla loppujakson ajan yli 5,3 MW, että hälytysraja ylittyisi.

5.4 Testiajo 4: Tasaisten hetkellisarvojen tapaus

Testiajossa 4 käytetään pienimmän neliösumman menetelmää. Lopputulos olisi ollut joka tapauksessa sama myös kumulatiivisella menetelmällä, koska hetkellisarvo on aina vakio 3 MW. Laskentatulokset eroavat selvästi aiemmista, koska hetkellisarvot ovat kaikki samoja. Jakson pituutena on kolme tuntia ja laskentavälinä 2 minuuttia. Mitattava muuttuja on SYS_CPU_TotalTime. Näytössä tarkkailtava väli on 0 MW – 10 MW.



Kuva 14: Kuvaaja tasaisten hetkellisarvojen tapauksessa

Kuvassa 14 on kuvaaja, jossa hetkellisarvo on tasaisesti 3 MW. Kuvasta nähdään, että teho kasvaa tasaisesti kumulatiivisesti ennustetta kohden, joka on siis sama kuin hetkellisarvo. Hälytysrajaksi on laitettu 3 MW, koska tässä tapauksessa kun

hetkellisarvot ovat samansuuruisia ja suuruus on tiedossa, tehon tarve on helposti ennustettavissa. Kuvaajasta näemme myös, että tasaisten hetkellisarvojen tapauksessa keskiarvo, arvio, ja ennuste, kaikki kulkevat tasaisesti hetkellisarvojen kohdalla.

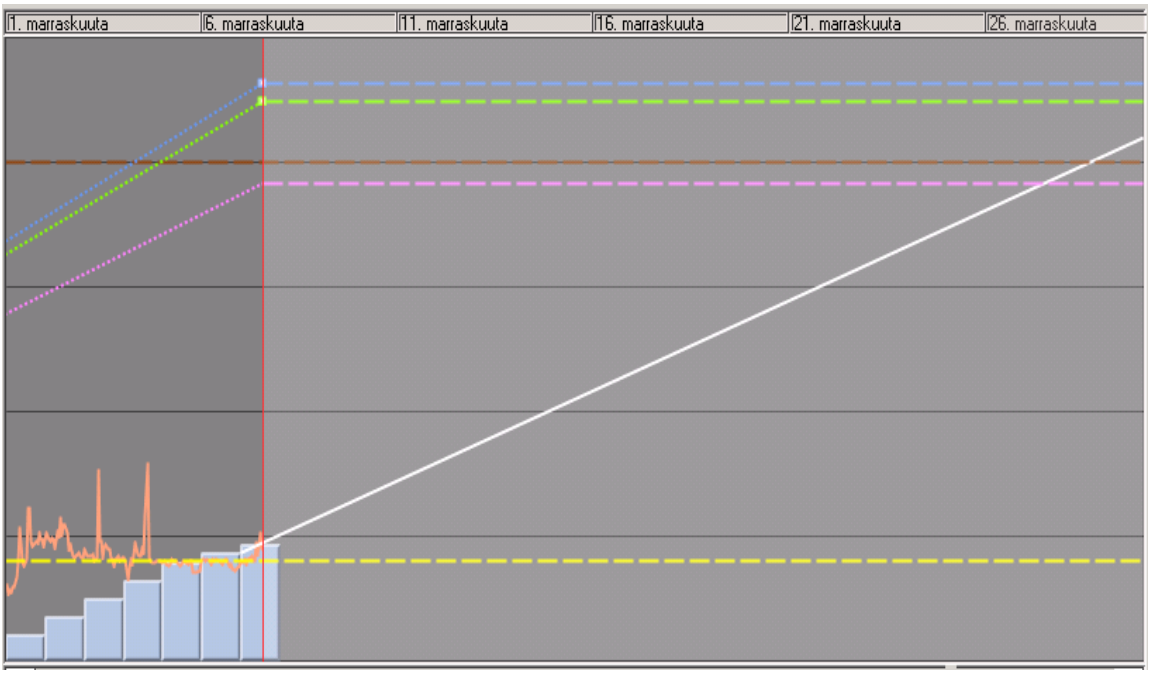
	Text	Current Value...	Unit (V...	5.10.2011 12:15:59	Average	Min Value	Max Value
	Tie-Line Power	3	%		3	3	3
	Cumulated Energy	1,8	MWh	0,3	0,9	0,1	1,8
	Projected Energy - line				2,4	1,8	3,0
	Projected Energy at Cycle End	3,0	MW				
	Plan	3,0	MW	3,0	3,0	3,0	3,0
	Plan2	10,0	MW	10,0	10,0	0,0	0,0
	Cycle Average Power	3,0	MW	2,7	2,9	2,7	3,0
	Time to Monitoring Limit	01:11:12	HH:mm...				
	Max Power to Cycle End	3,0	MW				
	Time to Cycle End	01:11:48	HH:mm...				
	EMO_EstimatePower_ACT (Pr...	3,0	MW	2,2	2,7	2,0	3,0
	EMO_Overshoot_ACT (Estima...	0,0	MW	-0,1	-0,2	-0,2	0,0

Kuva 15: Muuttujat ja mitattavat hetkellisarvot tasaisten hetkellisarvojen tapauksessa

Kuvassa 15 näemme mitattavat suureet sekä niiden hetkellisarvot, yksiköt, keskiarvot, sekä minimi- ja maksimiarvot. Aiemmissa kuvaajissa hetkellisarvot vaihtelivat rajusti, kun kuvan 15 esimerkissä taas hetkellisarvo on koko ajan sama 3 MW. Näemme, että hetkellisarvo, ennuste, keskiarvo, arvio ovat kaikki saman suuruisia. Koska koko ajan on sivuttu hälytysrajaa, maksimissaan voidaan kuluttaa saman verran tehoa pysyen vielä hälytysrajan sisällä. Tällä vauhdilla raja saavutetaan tunnin ja 11 minuutin päästä eli jakson loppuessa.

5.5 Testiajo 5: Kuukausikohtainen ennuste

Tehonvalvontasovelluksessa on myös mahdollista laskea historiapalkit ja ennuste koko kuukauden ajaksi. Testiajossa 5 jakson pituudeksi määriteltiin -1 eli yksi kuukausi. Laskentaväliksi määriteltiin 86400 sekuntia eli yksi vuorokausi. Yksi historiapalkki kuvaa siis aina yhtä vuorokautta. Kahden edellisen esimerkin tavoin myös esimerkkikoodissa 5 käytettiin uutta pienimmän neliösumman menetelmää ennusteen laskemiseen ja historiapalkkien sovittamiseen siihen.



Kuva 16: Kuukauden tehonkulutusta kuvaava kuvaaja

Kuvassa 16 näkyy kuvaaja, joka esittää tehonkulutusta yhden kuukauden osalta. Näemme yläpalkista, että kellonaikojen sijasta kuvaaja näyttääkin päivämääriä. Yksi palkki vastaa aina yhtä päivää. Ruskea viiva kuvaa kakkosrajaa, joka tässä esimerkissä ylitetään, toisin kuin aiemmissa testitapauksissa, joissa ruskea viiva ei ole edes näkynyt näytössä, vaan ollut niukasti sen yläpuolella. Näin siitäkin huolimatta, että koeajossa 5 raja kaksi on tuplaten niin suuri (20 MW) kuin aiemmissa testitapauksissa. Koska ensimmäinen raja (keltainen viiva) ylitetään reilusti, on hälytystä merkkaava violetti viiva hyvin korkealla.

Text	Current Value...	Unit (V...	1.11.2011 18:56:55	Average	Min Value	Max Value
Tie-Line Power	5,5	%	4,7	4,2	2,7	7,9
Cumulated Energy	4,3	MWh	1,1	3,0	1,1	4,7
Projected Energy - line				12,9	4,8	21,0
Projected Energy at Cycle End	23,1	MW				
Plan	4,0	MW	4,0	4,0	4,0	4,0
Plan2	20,0	MW	20,0	20,0	20,0	20,0
Cycle Average Power	22,4	MW	17,0	22,4	22,4	22,4
Time to Monitoring Limit	00:00:00	HH:mm...				
Max Power to Cycle End	-0,6	MW				
Time to Cycle End	10:06:37	HH:mm...				
EMO_EstimatePower_ACT_Month (Projected...	23,1	MW	17,6	23,1	23,1	23,1
EMO_Overshoot_ACT_Month (Estimated Ov...	19,1	MW	14,5	19,1	19,1	19,1

Kuva 17: Muuttujat ja hetkellisarvon kuukausikohtaisessa laskennassa

Kuvassa 17 näemme kuukausilaskennassa mitattavat suureet, niiden hetkellisarvot, yksiköt, keskiarvot, sekä minimi ja maksimiarvot. Esimerkiksi näemme hetkellisarvojen

vaihdelleen kuluneen jakson aikana 2,7 MW ja 7,9 MW välillä. Tämä on huomattavasti pienempi vaihtelu kuin aiemmissa testitapauksissa, mikä johtuu siitä, että nyt hetkellisarvot mitattiin tuntikohtaisista hetkellisarvoista raaka-arvojen sijaan. Hälytysraja on 4 MW. Tehonkulutuksen keskiarvo on 22,4 MW. Ennustettu tehonkulutus jakson lopussa on 23,1 MW. Hälytysraja on jo saavutettu, vaikka menossa on vasta seitsemäs päivä kuluvaa kuukautta. Arvioitu ylitys on 19,1 MW.

6 Jatkokehitysideoita

Tehonvalvontasovellusta kehittäessä tuli ilmi myös useita kehitysideoita tulevaisuutta varten. Sovellus ei vielä tullut täysin valmiiksi, vaan sovelluksen kehitystyö jatkuu ABB Oy:ssa. Tähän mennessä testejä on tehty vain yksittäisten ajokertojen perusteella. Seuraava vaihe on laittaa sovellus testikäyttöön pyörimään jatkuvasti, samaan tapaan kuin se tulee toimimaan oikeassakin käytössä. Tällä tavalla saadaan vastaavasti tutkittua jatkuvassa käytössä tehonvalvontasovelluksen toimintaa asiakasjärjestelmissä.

Tehonvalvontasovellukselle on myös tarkoitus tehdä omat määrittelytaulut EMOTieLineConfig ja EMOTielineConfigLimits. Näihin tulisi sitten myös vanhasta tietokantataulusta puuttuvat kentät uusien ominaisuuksien määrittelyä varten. Esimerkiksi hälytysrajojen määritykset tehtäisiin EMOTielineConfigLimits-tauluun, kun vielä nykyisessä versiossa ne kovakoodataan. Lisäksi on tarkoitus luoda päivityskomentotiedosto, jotta saadaan suoritettua kertaalleen tehtävä määrittelytaulun tietojen siirto uusiin määrittelytauluihin.

7 Yhteenveto

Insinöörityön tavoitteena oli luoda ABB Oy:lle uusi versio tehonvalvontasovelluksesta, joka on osa EM-järjestelmää. Tehonvalvontasovellus oli olemassa C++-ohjelmointikielelle tehtynä, mutta se haluttiin kääntää C#-ohjelmointikielelle ja samalla nykyaikaistaa ja uudistaa sovellusta.

Tehonvalvontasovellus saatiin käännetyksi kokonaisuudessaan ja uusia ominaisuuksia lisättiin. Uudessa versiossa käytettiin oliopohjaista lähestymistapaa. Yksi merkittävä lisätty ominaisuus on ennusteen laskeminen pienimmän neliösumman sovitteen mukaan, jonka tarkoituksena on näyttää tasaisia historiapylväitä sekä pyrkiä mahdollisimman tarkkaan ennustukseen ja pyrkimään vähäisempään reaktioon väliaikaisista muutoksista. Toinen keskeinen uusi ominaisuus on hälytysrajojen säätäminen kuluneen ajan, sekä ala- ja ylärajan mukaan, jotta turhilta hälytyksiltä vältyttäisiin.

Sovellusta tullaan vielä parantamaan ennen julkaisua lisäämällä siihen jatkokehitysideoissa mainittuja ominaisuuksia. Sovellusta ei myöskään ole testattu tarpeeksi kattavasti eikä jatkuvassa käytössä ja testejä tullaan tekemään ennen käyttöönottoa. Tämän jälkeen näillä näkymillä sovellus tullaan liittämään osaksi EM-järjestelmää ja se tullaan ottamaan käyttöön ABB Oy:n asiakasyrityksissä.

Lähteet

1 Kymäläinen, Toni. cmpPlus Energy Manager User's Reference Manual. 2009. PDF-dokumentti. ABB Oy.

2 Mäntysaari, Juha. cpmPlus Energy Manager (EM). 2010. PDF-dokumentti. ABB Oy.

3 cpmPlus Energy Manager Overview. 2010. PDF-dokumentti. ABB Oy.

4 Loisteho . 2011. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Loisteho>.Luettu 26.09.2011.

5 Pätöteho. 2011. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Loisteho>.Luettu 30.09.2011.

6 Sähkö. Verkkodokumentti . <http://personal.inet.fi/private/procyon/pub/Sahkooppi.pdf> . Luettu 30.09.2011.

7 Kulmala, Jari. Industrial IT Real Time DataBase Developer's Manual .2007. PDF-dokumentti. ABB Oy.

8 Saari, Juho. Industrial IT User Interface User's Reference Manual. 2006. PDF-dokumentti. ABB Oy.

9 C Sharp. 2011. Verkkodokumentti.Wikipedia. <http://fi.wikipedia.org/wiki/C_sharp> . Luettu 3.10.2011.

10 C# Language Specification. 2006. Verkkodokumentti. Ecma-international. <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf. Luettu 3.10.2011.

11 Why Microsoft's C# isn't. 2002. Verkkodokumentti.Cnet News. <http://news.cnet.com/2008-1082-817522.html> . Luettu 3.10.2011.

12 Microsoft's blind spot. 2002. Verkkodokumentti.Cnet News. <http://news.cnet.com/2010-1071-831385.html> . Luettu 3.10.2011.

13 Syrjälä, Santtu. C#-pikakurssi.2005. Verkkodokumentti. Jyväskylän yliopisto <http://users.jyu.fi/~vesal/kurssit/winohj/csharp/seminaari05/CS_Pikakurssi.pdf>. Luettu 22.09.2011.

14 Boxing and Unboxing (C# Programming Guide). 2010. Verkkodokumentti. Microsoft. <http://msdn.microsoft.com/en-us/library/yz2be5wk.aspx>. Luettu 25.10.2011.

15 Osborn, John. Deep Inside C#: An Interview with Microsoft Chief Architect Anders Hejsberg. 2000 . Verkkodokumentti. O'Reilly Windows DevCenter.

<http://www.windowsdevcenter.com/lpt/a/2273>Luettu 26.09.2011.

16 Current local time in Helsinki . Verkkodokumentti. timeanddate.com
< http://www.timeanddate.com/worldclock/city.html?n=101 >.Luettu 4.11.2011.

17 Kolari, Mika. C#/.NET – Aika ja päivämäärä. Verkkodokumentti. mikakolari.fi
<http://mikakolari.fi/csharp-dotnet/net-luokkakirjastoja/aika-ja-paivamaara/>.Luettu 22.09.2011.

18 Cumulative Sum. Verkkodokumentti. WolframMathWorld
<http://mathworld.wolfram.com/CumulativeSum.html>.Luettu 22.09.2011.

19 Parker, Nick. make pair in C#. 2004. Verkkodokumentti. Developer Notes
<http://developernotes.com/archive/2004/11/23/make_pair-in-c.aspx>.Luettu 22.09.2011.

20 Least squares.2011. Verkkodokumentti. Wikipedia.
<http://en.wikipedia.org/wiki/Least_squares>.Luettu 20.09.2011.

21 Teikari, Ismo. Pienimmän neliösumman menetelmä . 2004. Verkkodokumentti. Tilastokeskus.
<http://193.166.173.45/tup/tietoaika/tilaajat/ta_11_04_teikari.html>.Luettu 23.09.2011.

Esimerkkikoodi 2: Historiatieto- ja ennustetaulujen kirjoitus tietokantaan kokonaan

```
try
{
 System.DateTime time = start;
 mDriver.Variables.Request(CumulatedVarIDN);
 BB.Vtrin.cDbVariable var = mDriver.Variables[CumulatedVarIDN];
 if (var == null)
 {
  return false;
 }
 mDriver.Histories.Request(forCurrentHistory);
 ABB.Vtrin.cDbHistory hist =
mDriver.Histories[forCurrentHistory];
 if (hist == null)
 {
  var = null; LogMessage("Error while writing datas.");
  return false;
 }

 ABB.Vtrin.Drivers.cGraphFetchParameters prms =
ABB.Vtrin.Drivers.cGraphFetchParameters.CreateScalarFetch(
       mDriver.Classes["ProcessHistory"],
       mBaseTime,
       "Variable=? AND History=?",
       var.Id,
       hist.Id);
 LogMessage("Data added to database");
 ABB.Vtrin.Util.IGraphDataIterator gdi =
mDriver.GetGraphDataIterator(mDriver.Classes["ProcessHistory"],
prms);
 if (gdi != null)
 {
  if (mFormula.Equals("Least squares"))
  {
   for (int j = 0; j < data.Length; j++)
   {
    if (time >= start && time < end) gdi.Write(time,
mLeastSquaresTable[j], ABB.Vtrin.cValueStatus.OK |
ABB.Vtrin.cValueStatus.Representativeness);
    time = time.AddSeconds(IntervalSeconds);
   }
   for (int j = 0; j < futureData.Length - 1; j++)
   {
    if (j == futureData.Length - 2)
    {
     time = time.AddTicks(-1);
    }
```

```
     gdi.Write(time, (object)futureData[j],
ABB.Vtrin.cValueStatus.OK |
ABB.Vtrin.cValueStatus.Representativeness);
     time = time.AddSeconds(IntervalSeconds);
    }
   }
   else
   {
    for (int j = 0; j < data.Length; j++)
    {
     gdi.Write(time, (object)data[j], ABB.Vtrin.cValueStatus.OK |
ABB.Vtrin.cValueStatus.Representativeness);
     time = time.AddSeconds(IntervalSeconds);
    }
    //time = time.AddSeconds(IntervalSeconds*2);
    for (int j = 0; j < futureData.Length; j++)
    {
     if (j == futureData.Length - 1)
     {
      time = time.AddTicks(-1);
     }
     gdi.Write(time, (object)futureData[j],
ABB.Vtrin.cValueStatus.OK |
ABB.Vtrin.cValueStatus.Representativeness);
     time = time.AddSeconds(IntervalSeconds);
    }
   }
  }
  else
  {
   var = null;
   prms = null;
   hist = null;
   gdi.Close();

   return false;
  }
```